SICS Technical Report
T99:05

ISRN: SICS-T--99/05-SE
ISSN: 1100-3154

Traffic measurement and analysis *

Henrik Abrahamsson

September 1999

henrik@sics.se

Swedish Institute of Computer Science
Box 1263, SE-164 29 Kista, Sweden

Abstract

Measurement and analysis of real traffic is important to gain knowledge about the characteristics of the traffic. Without measurement, it is impossible to build realistic traffic models. It is recent that data traffic was found to have self-similar properties. In this thesis work traffic captured on the network at SICS and on the Supernet, is shown to have this fractal-like behaviour. The traffic is also examined with respect to which protocols and packet sizes are present and in what proportions. In the SICS trace most packets are small, TCP is shown to be the predominant transport protocol and NNTP the most common application. In contrast to this, large UDP packets sent between not well-known ports dominates the Supernet traffic. Finally, characteristics of the client side of the WWW traffic are examined more closely. In order to extract useful information from the packet trace, web browsers use of TCP and HTTP is investigated including new features in HTTP/1.1 such as persistent connections and pipelining. Empirical probability distributions are derived describing session lengths, time between user clicks and the amount of data transferred due to a single user click. These probability distributions make up a simple model of WWW-sessions.

Keywords: Traffic measurement, self-similarity.

* This work was supported by Telia AB.

Examensarbete MN3
1999-09-15

Traffic measurement and analysis

Henrik Abrahamsson

Information Technology
Department of Computer Systems
Uppsala University
Box 325
SE-751 05 Uppsala
Sweden

This work has been carried out at
SICS
Box 1263
SE-164 29 Kista
Sweden

Supervisor: Bengt Ahlgren
Examiner: Jakob Carlström

Contents

Abstract

Measurement and analysis of real traffic is important to gain knowledge about the characteristics of the traffic. Without measurement, it is impossible to build realistic traffic models. It is recent [18] that data traffic was found to have self-similar properties. In this thesis work traffic captured on the network at SICS and on the Supernet, is shown to have this fractal-like behaviour. The traffic is also examined with respect to which protocols and packet sizes are present and in what proportions. In the SICS trace most packets are small, TCP is shown to be the predominant transport protocol and NNTP the most common application. In contrast to this, large UDP packets sent between not well-known ports dominates the Supernet traffic. Finally, characteristics of the client side of the WWW traffic are examined more closely. In order to extract useful information from the packet trace, web browsers use of TCP and HTTP is investigated including new features in HTTP/1.1 such as persistent connections and pipelining. Empirical probability distributions are derived describing session lengths, time between user clicks and the amount of data transferred due to a single user click. These probability distributions make up a simple model of WWW-sessions.

1.0 Introduction

Measurement and analysis of real traffic is important to gain knowledge about the characteristics of the traffic. Without measurement, it is impossible to build realistic theoretical traffic models. The traditional telephone network could very successfully be analysed and modelled using applied mathematics such as stochastic processes. Especially Poisson processes have been used which states that call arrivals are mutually independent and that the call interarrival times are all exponentially distributed, with one and the same parameter λ. Because of the success of voice network modelling and because Poisson processes have some attractive theoretical properties, the same approach have often been used when modelling data network traffic. Packet and connection arrivals have been assumed to be Poisson processes. But several studies [21] have shown that the distribution of packet interarrivals clearly differs from exponential and Leland *et al*. [18] showed that the burstiness on many timescales, observed in real traffic, can not be described with traditional Poisson-based traffic modelling. Instead they introduced statistically *self-similar processes* as a better way of modelling LAN traffic.

In this thesis work, network traffic on the Supernet and external traffic at SICS is analysed. The traffic was captured using *tcpdump* [15]. Figure 1 shows the network at SICS. The machine running *tcpdump* (called *network monitor* in the figure) was listening to the 100 Mbit/sec line connecting all workstations at SICS with the gateway and in the end the SUNET network. This was used to capture all conversations between machines at SICS and the outside Internet world, for 24 hours. The packet trace was taken between 21:36 990414 and 21:40 990415 and includes more than 21 million packets.

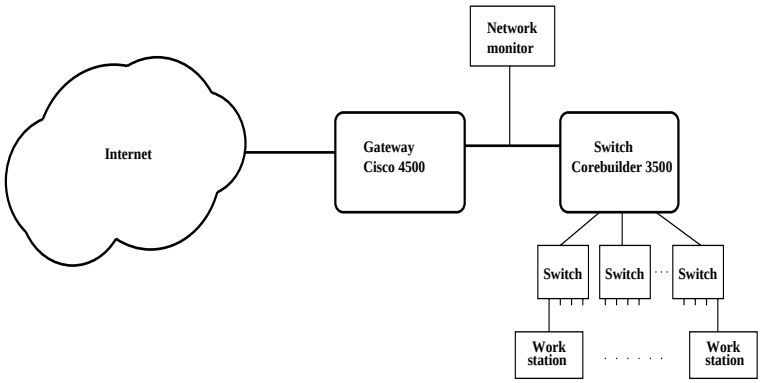

Figure 1. The network at SICS.

The Supernet trace includes 15 million packets captured between 15:27 and 23:06 980423 on the Supernet in Sundsvall. Supernet consists of a mixture of Ethernet, ADSL (Asymmetric Digital Subscriber Line) and cable TV modems. Figure 2 shows the part of the network where the trace was taken. The trace includes all traffic between the ADSL-switch and the router and also the traffic between the two switches. ADSL provides approximately 7.6 Mbit/sec of downstream bandwidth (to the costumer) and 1.8 Mbit/sec of return bandwidth. For the cable modems each outgoing line from the switch makes up a segment where the active modems share 10 Mbit/sec of bandwidth.

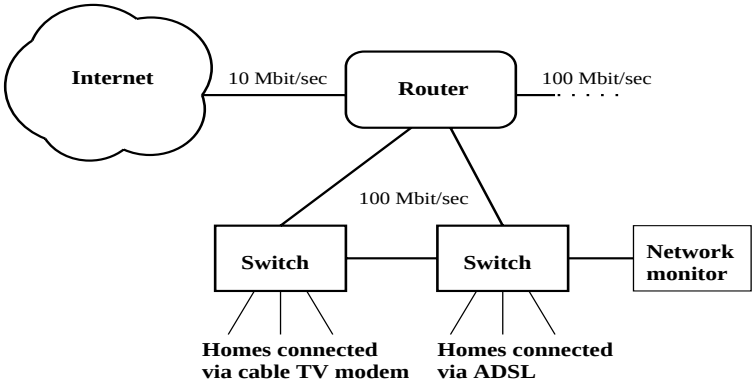

Figure 2. Supernet.

The work is organized as follows. In Section 2 these packet traces are examined with respect to self-similarity. The section begins with definitions and a description of the methods used for estimating the degree of self-similarity. The result of applying these methods to the SICS and Supernet traffic data is presented in 2.3, and the section concludes with a discussion of what the possible causes and implications of self-similarity might be. In Section 3 the traffic is examined with respect to which protocols and packet sizes are present and in what proportions. To put the results in context this section begins with a brief outline of the TCP/IP protocol suite. The results show for instance the packet size distribution, the composition of the IP traffic during the time the traces were taken, which transport protocol is most common and what applications are the most popular. Finally in Section 4, characteristics of the client side of the WWW traffic are examined more closely. In order to extract useful information from the packet trace, web browsers use of TCP and HTTP is investigated including new

features in HTTP/1.1 such as persistent connections and pipelining. Empirical probability distributions are derived describing session lengths, time between user clicks and the amount of data transferred due to a single user click. These probability distributions make up a simple model of WWW-sessions.

2.0 Self-similarity

In Section 2.1 the mathematics used are presented, leading up to the definition of self-similarity. Section 2.2 deals with the methods used for estimating the Hurst parameter that describes the degree of self-similarity. In Section 2.3 the result of applying these methods to the SICS and Supernet traces are presented, and in 2.4 these results are put in context when possible causes and implications of self-similarity are discussed.

2.1 Definitions

A *random variable* is a quantity that each time it is measured takes on one of a range of values. Particular values occur with different probabilities. Each separate measurement is referred to as an *instance* of the random variable. A generic random variable is denoted X, and x_i represents the ith instance of X. Unless otherwise stated, it is assumed that there are a total of n instances. X is *discrete* if it assumes a finite or countable number of values. The random variable is *continuous* if it assumes all values in an interval according to a density function $f_X(x)$.

The *Cumulative Distribution Function (CDF)* of a random variable X tells the probability that an instance of X is less than or equal to a given value x.

$$F_X(x) = P(X \leq x), \quad -\infty < x < \infty$$

The derivative of the *CDF* is called the *probability density function (pdf)* of X:

$$f_X(x) = \frac{d}{dx} F_X(x)$$

If X is discrete the CDF is not differentiable, and instead of the pdf the *probability function* $p_x(k) = P(X=k)$, (k = 0,1,....) is used. This function tells the probability that an instance of X is equal to a given value x. Some well-known distributions mentioned later on are:

$$\textit{Poisson distribution: } p_X(k) = e^{-m} m^k / k! \quad (k=0,1,...),$$

$$\textit{Exponential distribution: } f_X(x) = \frac{1}{m} e^{-x/m} \quad x \geq 0 \quad \text{and}$$

$$\textit{Pareto distribution: } F_X(x) = P(X \leq x) = 1 - (\alpha / x)^{\beta} \quad \alpha, \beta \geq 0, \; x \geq \alpha.$$

If in the Pareto distribution $\beta \leq 2$, then the distribution has infinite variance, and if $\beta \leq 1$ then it has infinite mean. The Pareto distribution is an example of a *heavy-tailed* distribution. A distribution is said to be *heavy-tailed* if

$$P(X \geq x) \sim cx^{-\beta}, \text{ as } x \to \infty, \beta \geq 0.$$

In practical terms a random variable that follows a *heavy-tailed* distribution can give rise to extremely large values with non-negligible probability.

The cumulative distribution function or the density/probability function is used to get a complete description of a random variable. To get a description in more condensed form *indices of central tendencies and dispersion* is often used. The purpose of an index of central tendency is to summarize the data by a single number that (to be meaningful) should be representative of the major part of the data set. The most common used is the *mean* or *expected value*

$$E(X) = \mu = \sum_i x_i p(x_i) = \int_{-\infty}^{\infty} xf(x)dx,$$

where summation is used for discrete variables and integration for continuous random variables. An alternative is the *median,* which is obtained by sorting the observations in an increasing order and taking the observation that is in the middle of the series. To avoid drowning crossing a stream with an average depth of six inches, it is also important to know something about the indices of dispersion. These specifies the variability in a data set. Three popular alternatives are:

$$\textit{variance: } V(X) = \sigma^2 = E[(X-\mu)^2]$$

$$\textit{standard deviation: } D(X) = \sigma = \sqrt{V(X)}$$

$$\textit{coefficient of variation}: R(X) = \frac{D(X)}{E(X)}.$$

Often it is also important to know if two random variables are dependent of each other. This can be examined using the simultaneous probability- or density function. But it is more often expressed as a single number using the *covariance* or the *correlation coefficient*. Given two random variables *X* and *Y* with means μ_x and μ_y, their *covariance* is

$$Cov(X, Y) = E[(X-\mu_x)(Y-\mu_y)].$$

For independent variables, the covariance is zero. But the reverse is not true, it is possible for two variables to be dependent and still have zero covariance.The other measure of dependence between two random variables is the *correlation coefficient*:

$$\rho(X, Y) = \frac{Cov(X, Y)}{D(X)D(Y)}$$

The correlation coefficient always lies between -1 and 1.

When data is collected sequentially in time, a *time series* can be used for modelling and predictions. A *time series* [2] is a set of observations x_t, each one being recorded at a specified time *t.* In a *discrete-time series* the set of times in which observations are made is a discrete set, as is the case when observations are made at fixed time intervals. *Continuous-time series* are obtained when observations are made continuously over some time interval. The observations x_t are often supposed to be instances of a random variable *X*, and the time series is modelled as a *stochastic process.* A stochastic process is a family of random variables $\{X(t), t \in T\}$ with the same range. If *T* is an interval of real numbers then the process is said to have continuous time, and if *T* is a sequence of integers it is said to have discrete time. The term *time series* is often used to mean both the data and the process of which it is a realization.

The autocovariance function of a process $\{X(t), t \in T\}$ with finite variance is defined by $\gamma_X(r, s) = Cov(X(r), X(s)) \quad r, s \in T$. A discrete time series is said to be *covariance stationary* (or *weak stationary* or sometimes just *stationary*) if the expected value of X_t is finite and equal to the same value *m* for all *t*, and it holds that $\gamma_X(r, s) = \gamma_X(r + t, s + t)$ for all *r, s*, and *t*. That is, the series is covariance stationary if the mean is the same all the time and the dependence between all equally distanced pairs of observation is the same.

The autocovariance function can be redefined for a stationary process as a function of just one variable: $\gamma_X(h) = \gamma_X(h, 0) = Cov(X(t + h), X(t))$ for all *t* and *h*. The *autocorrelation function* of $\{X_t\}$ is defined analogously as the function whose value at lag *h* is $\rho_X(h) = \gamma_X(h)/\gamma_X(0) = \rho(X(t + h), X(t))$ for all *t* and *h*.

If a series is *strict stationary* then X_t has the same distribution for all *t*, which implies that the expected value and variance are constant and the covariance is the same for all *h*. A process is said to have *stationary increments* if the distribution of *X(t+h)-X(t)* only depends on *h*.

For a detailed discussion of *self-similarity* and *long-range dependence* see Beran [4] and Willinger *et al.*[31], [33]. The description in this subsection follows those sources closely. There are a number of different, not equivalent, definitions of *self-similarity.* The standard one states that a continuous-time process $Y = \{Y(t), t \geq 0\}$ is self-similar with self-similar parameter *H* if it satisfies the condition:

$$Y(t) \overset{d}{=} a^{-H} Y(at) \quad t \geq 0, a > 0, 0 < H < 1$$

where the equality is in the sense of finite-dimensional distributions. While a process *Y* satisfying this can never be stationary, that would require $Y(t) \overset{d}{=} Y(at)$, *Y* is typically assumed to have stationary increments. A second definition of self-similarity that is more appropriate in the context of standard time series, involves a stationary sequence $X = \{X(i), i \geq 1\}$. Let

$$X^{(m)}(k) = (1/m) \sum_{i = (k-1)m + 1}^{km} X(i), \; k = 1,2,...,$$

be the corresponding aggregated sequence with level of aggregation *m*, obtained by dividing the original series *X* into blocks of size *m* and averaging over each block. The index *k* labels the block. If *X* is the incremented process of a self-similar process *Y*, that is *X(i) = Y(i+1) - Y(i)*, then for all integers *m*,

$$X \overset{d}{=} m^{1-H} X^{(m)}.$$

If a stationary sequence $X = \{X(i), i \geq 1\}$ satisfies this for all aggregation levels *m*, then it is called *exactly self-similar.* It is said to be *asymptotically self-similar* if it holds as $m \to \infty$. Similarly, a covariance-stationary sequence *X(i),* $i \geq 1$ is called *exactly second-order self-similar* if $m^{1-H} X^{(m)}$ has the same variance and autocorrelation as *X* for all *m*. It is said to be *asymptotically second-order self-similar* if it holds as $m \to \infty$.

A related notion is that of *long-range dependence* (LRD), which means correlations across large time scales. A stationary process is long-range dependent if its autocorrelation function $\rho_X(h)$ is nonsummable:

$$\sum_{h=1}^{\infty} \rho_X(h) = \infty$$

Thus the definition of long-range dependence applies only to infinite time series. The two notions of long-range dependence and self-similarity are in general not equivalent. Long-range dependence is one of the ways in which self-similarity manifests itself [18] and self-similar processes are the simplest models with long-range dependence [21]. Self-similarity typically refers to scaling behaviour of the distributions of a continuous or discrete time process, while long-range dependence involves the tail behaviour of the autocorrelation function of a stationary time series. But since second-order self-similar also is defined in terms of autocorrelations, the terms long-range dependence and (exactly or asymptotically second-order) self-similarity are sometimes used in an interchangeable fashion, because both refers to the tail behaviour of the autocorrelations and are essentially equivalent [31].

One attractive feature with self-similar models is that the degree of self-similarity is expressed using only a single parameter, the so called *Hurst parameter H*. For self-similar series with long-range dependence, $1/2 < H < 1$, and as $H \to 1$ the degree of both self-similarity and long-range dependence increases.

2.2 Methods for estimating the Hurst parameter

It is not possible to use the definition to check whether a finite traffic trace is self-similar or not. Instead different features of self-similarity such as slowly decaying variances are investigated in order to estimate the Hurst parameter *H*. As mentioned in section 2.1, this parameter *H* can take any value between 1/2 and 1 and the higher the value the higher the degree of self-similarity. For smooth Poisson traffic the value is *H*=0.5. Here four methods are used to test for self-similarity. These four methods are all heuristic graphical methods, they provide no confidence intervals and they may be biased for some values of *H*. The *rescaled adjusted range plot* (R/S plot), the *Variance-Time plot*

and the *Periodogram plot*, and also the theory behind these methods, are described in detail by Beran [4] and Taqqu *et al*. [28]. Molnar *et al.* [20] describes the *index of dispersion for counts* method and also discuss how the estimation of the Hurst parameter can depend on estimation technique, sample size, time scale and other factors.

Each packet captured with *tcpdump* has a timestamp. The traffic trace is divided up into time intervals (bins), for instance of size 100 ms. For each time interval the number of packets or bytes that arrived is counted. The resulting vector, with the number of packets (or bytes) that arrived in each time interval, is input to all four of the methods described.

2.2.1 The R/S method

The *R/S* method is one of the oldest and most well known methods for estimating *H*. Let X_t denote the number of packets that arrive at time *t*, i.e the number of packets in bin *t*, and let

$$Y_j = \sum_{i=1}^{j} X_i$$

be the cumulative inflow up to time *j*. The *R/S*-statistic or rescaled adjusted range is defined by the ratio

$$R/S = \frac{R(t,k)}{S(t,k)} \qquad \text{where}$$

$$R(t,k) = \max_{0 \le i \le k} \left[Y_{t+i} - Y_t - \frac{i}{k}(Y_{t+k} - Y_t) \right] - \min_{0 \le i \le k} \left[Y_{t+i} - Y_t - \frac{i}{k}(Y_{t+k} - Y_t) \right]$$

is called the adjusted range and

$$S(t,k) = \sqrt{k^{-1} \sum_{i=t+1}^{t+k} (X_i - \overline{X}_{t,k})^2} \quad \text{where} \quad \overline{X}_{t,k} = k^{-1} \sum_{i=t+1}^{t+k} X_i$$

makes it possible to study properties that are independent of scale.

To determine the Hurst parameter *H* the ratio *R/S* is calculated for every possible, or a sufficient number of, values of *t* and *k* and log *R/S* is plotted against log *k*. The slope of a straight line fitted to the points in the plot, for instance by the least square method, is an estimation of the parameter *H*.

In practice the ratio *R/S* is not calculated for every possible *t* and *k*. Instead a number of equally spaced starting points *t* and a number of intervals (lags) *k* are chosen. Typically logarithmically spaced values of *k* is chosen because log *R/S* is to be plotted versus log *k*. For each starting point *t* the ratio *R/S* is calculated for every lag *k* such that *t+k* <= length of *X*. For small *k* one get many estimates of *R/S* but for large *k* one gets only a few, down to one, estimate of *R/S*.

The *R/S* method is known [28] to be biased towards *H*=0.7. It is biased upwards for small values of *H* and downwards for large values of *H*.

A number of questions arise: The ratio *R/S* can not in practice be calculated for every possible starting point *t* and lag *k*. What is *a sufficient number* of different values of *t* and *k*? The low and high ends of the plot is usually not used when estimating *H* because of the influence from short-range dependence in the low end and because there are too few points to make a reliable estimate in the high end. How to choose cut-offs? It turns out that slight changes in the values of the cut-offs and in the number of values of *t* and *k* don't affect the estimate very much. The values used, when estimating *H* for the SICS and Supernet traces, are described in 2.2.5.

2.2.2 Variance-Time plot

Let *X* be a vector with the number of packets in each interval (bin). If for example the bin size has been chosen to 100 ms then X_1 is the number of packets that arrived the first 100 ms. Characteristic of long-range dependent processes is that the variance of the sample mean converges slower to zero than 1/*n* (the reciprocal of the sample size). It can be shown that

$$var(\overline{X}_n) \approx cn^{2H-2}$$

where $c > 0$.

This is what the *variance-time plot* method is based on and the actual method to estimate *H* is as follows:

First the mean of each pair of consecutive, non-overlapping bins are calculated and then the variance of these means is calculated. The 2-logarithm of the variance is plotted against the logarithm of the block size i.e 1. Then the same thing is done for blocks of size 4,8,16,..,length(*X*)/2 bins. The parameter *H* can be estimated by fitting a simple least squares line through the resulting points and using the relation *slope* = 2*H* - 2.

The values for the smallest and largest block sizes are usually not included when estimating H. The problem is the same as for the *R/S* method. How to choose cut-offs?

2.2.3 The Periodogram method

The periodogram is defined as

$$I(\upsilon) = \frac{1}{2\pi N}\left|\sum_{j=1}^{N} X(j)e^{ij\nu}\right|^2$$

where υ is a frequency, N is the length of the series, and X is the time series. I(υ) is an estimator of the spectral density. A series with long-range dependence should have a periodogram which is proportional to $|\upsilon|^{1-2H}$ close to the origin. An estimation of H is given by fitting a straight line to a log-log plot of the periodogram against the frequency. The slope of the line is approximately 1-2*H*. In practice only the lowest 10% of the roughly N/2 frequencies is used when estimating *H* [27].

2.2.4 Index of Dispersion for Counts

The *index of dispersion for counts* is a common used measure for capturing the variability of traffic over different time scales [18], [20]. For a given time interval *t* the index of dispersion for counts (IDC) is given by the variance of the number of arrivals during the interval divided by the expected value. The vector X with the number of packets that arrived in each interval (bin) is divided into non-overlapping blocks of length *t*. *IDC(t)* is the variance of the number of packets in the blocks divided by the mean. *IDC(t)* is calculated for increasing block sizes *t* and for self-similar processes the values increases monotonically. The Hurst parameter *H* can be estimated by plotting *log IDC(t)* against *log t* which results in an asymtotic straight line with slope *2H-1* [18].

To get a reliable estimate of *IDC(t)* the maximum block size is limited to 10% of the sample size [20]. Using non-overlapping blocks of length *t* at least about 10 values is needed to calculate the variance with acceptable confidence. Thus the calculated *IDC(t)* value is getting more and more inaccurate as *t* increases. As a result, the IDC plot becomes more and more noisy as *t* increases.

2.2.5 Implementation

The methods described for estimating the Hurst parameter were implemented using *Matlab.* In Section 2.3, the results of applying these methods to the SICS and Supernet traces are presented. But first the implementations were tested on traces, taken at the Bellcore Morristown Research and Engineering Center, which are available at the Internet Traffic Archive [14]. These traces are a subset of those analysed in Leland *et al*. [18]. The estimates of the Hurst parameter that were obtained by applying the methods to the Bellcore data were compared to the results presented in [18] in order to confirm that the implementations give reasonable results. For the R/S method different number of starting points *t* and lags *k* and different cut-offs were tried. Since the results were almost the same the method seems to be stable and when the results are presented in Section 2.3 the same values of these parameters are used. Twenty starting points were used and the lower cut-off was chosen as in Taqqu *et al*. [28] to 10^0.7. Besides this minimum, lags of sizes 10^0.75, 10^0.8,...,length(sample) were used. For the Variance-Time method the problem is to decide which of the plotted points should be included when determining the slope. The points due to the smallest and largest block sizes were not used when estimating the Hurst parameter. For the IDC method the maximum block size was restricted to 10% of the sample size when estimating H. The implementation of the Periodogram method seemed to work well for the traffic in the SICS trace when the proposed [27] lower 10% of the frequencies were used. But for the Supernet traffic and the Bellcore data the method sometimes gives absurd results. Therefore one Periodogram plot is shown for the SICS trace in 2.3.1, but all other results presented relies only on the R/S, Variance-Time and IDC methods.

2.3 Is SICS and Supernet traffic self-similar?

The methods described in 2.2 for estimating the Hurst parameter were used to investigate if the traffic captured in the SICS and Supernet traces are self-similar. The definitions of self-similarity and long-range dependence rely on the fact that the traffic, or the stochastic process describing the traffic, is stationary (or have stationary increments). This means for instance that the expected number of packets that arrive in 100 ms

should be the same irrespective of when the traffic is investigated. This would not be the case if 24 hour of traffic was examined at once, since the number of users and the load on the network varies during the day. The fact that it is not possible to tell with certainty whether or not the traffic is stationary and since it is difficult to distinguish stationary traffic with long-range dependence from certain non-stationary traffic with short-range dependence [13], at most one hour at a time of the traffic traces was examined.

2.3.1 SICS

The hour between 14:00 and 15:00 of the trace was chosen for closer examination. This choice was somewhat arbitrary but it seems to be a “normal hour” of traffic (see also Figure 11 in Section 3.2). First the number of packets that arrived each 100 ms interval was counted and Figure 3 shows the resulting R/S plot, Variance-Time plot, IDC and Periodogram plots. With the R/S method the Hurst parameter was estimated to H = 0.85. The line fitted by least-square to the Variance-Time plot has slope -0.386 which gives an estimate of H=1+slope/2 = 0.81. The Index of Dispersion for Counts estimates H to 0.77 and the Periodogram gives the value $H=0.84$. The methods don’t give exactly the same result but the values are all clearly above 0.5, so the traffic is self-similar with Hurst parameter $H \approx 0.8$.

Also, the byte traffic is self-similar with approximately the same H value. The number of bytes arriving in each time interval was counted and the R/S, Variance-Time and IDC methods were applied to this data. The results are shown in Table 1. This table also shows the resulting estimates of the Hurst parameter when different time intervals (bin sizes) were used, ranging from 10 ms to 1 second.

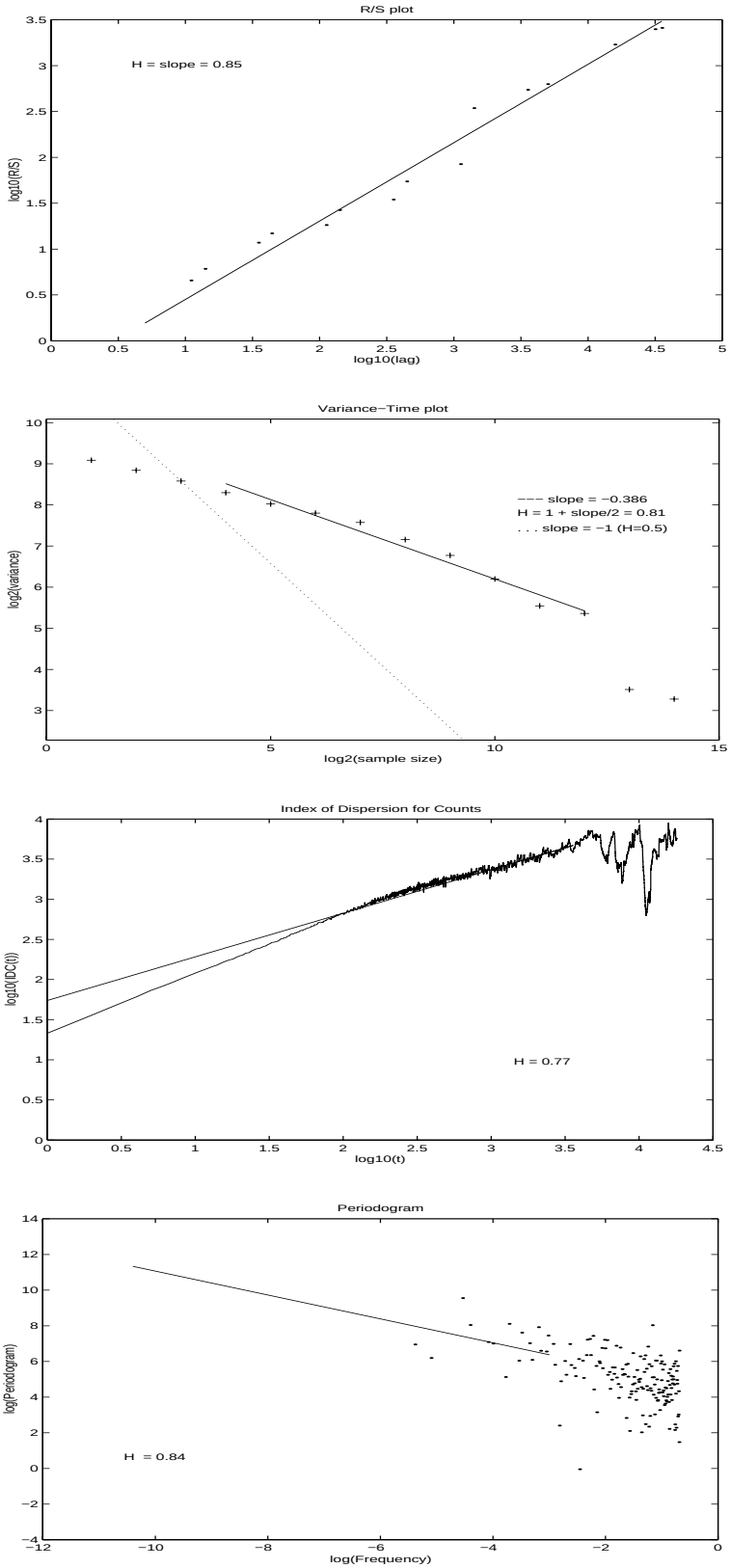

Figure 3. Estimates of the Hurst parameter H for the hour 14:00 -15:00 of the SICS trace. At the top the R/S plot followed by Variance-Time plot, Index of Dispersion for Counts and at the bottom the Periodogram plot.

Bin size:	Packets			Bytes		
	H_{rs}	H_{var}	H_{idc}	H_{rs}	H_{var}	H_{idc}
0.01 s	0.85	0.85	0.77	0.83	0.86	0.78
0.1 s	0.85	0.81	0.77	0.85	0.82	0.78
1 s	0.86	0.76	0.77	0.87	0.78	0.78

Table 1. Hurst parameter estimates for the hour 14:00-15:00 of the SICS trace.

With the fact in mind that non-stationary traffic can be mistaken for self-similar stationary traffic, even smaller parts of the trace was examined. The Hurst parameter was estimated for each of the six non-overlapping 10 minutes intervals between 14:00 and 15:00. The result is shown in Figure 4. None of the estimates is less than 0.75, so this hour of traffic is clearly self-similar.

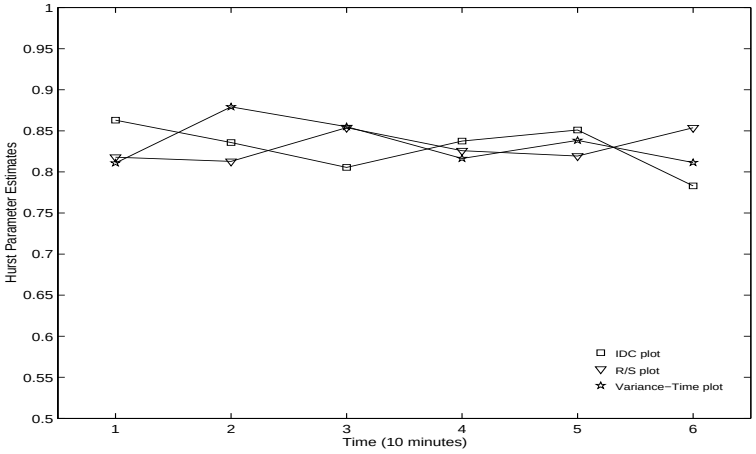

Figure 4. Hurst parameter estimates for ten minutes intervals 14:00 -15:00.

So far only one hour out of the 24 hours of traffic captured in the SICS trace has been analysed. The Hurst parameter was estimated for each hour of the trace and the results are presented in Figure 5. The estimates are for packet traffic and the bin size 100 ms was used. All three methods show high values between hour 12 and 19. The trace started at 21:36 so that means between 8:30 and 16:30. Since this is also the time of the day when most people use the network it corresponds well with the findings of Leland *et al*. [18] that the higher the load on the network the higher the degree of self-similarity.

It is also notable that the three methods used sometimes give quite varying estimates of the Hurst parameter. Especially for the fifth hour between 01:36 and 02:36 when the R/S method estimates *H* to approximately 0.75, the Variance-Time plot gives a much lower value and the IDC method states that *H*=0.5, thus not self-similar at all.

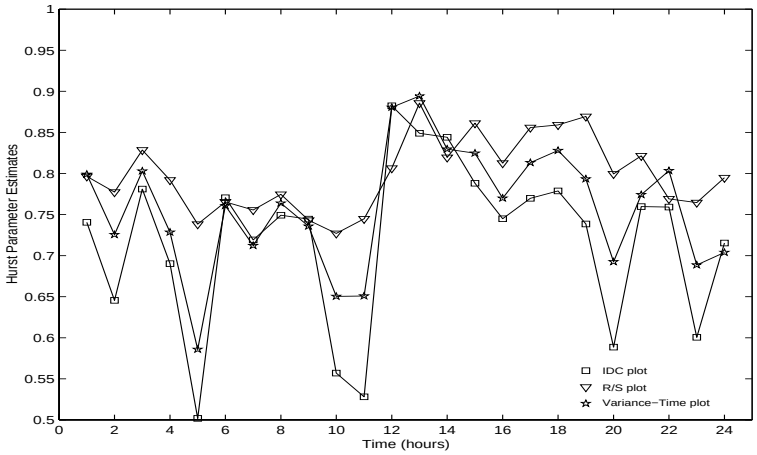

Figure 5. Hurst parameter estimates for each hour of the SICS trace.

2.3.2 Supernet

The Supernet trace consists of 150 different files, each containing 100000 packets. A single file covers between two and three minutes of network traffic but there is always a short silence, were packets are missing, before the next file starts. These intervals of silence between files ranges from 150 to 580 ms. It is hard to know how much impact these silences have on the estimate of the Hurst parameter, but they don't make the traffic less bursty. The hour between 19:42 and 20:42 was selected for a closer analysis since the silences between files were smallest during this hour.

The procedure is the same as for the SICS trace in section 2.3.1. The hour 19:42-20:42 was analysed in detail and the Hurst parameter was estimated using R/S plot, Variance-Time and IDC plots. The results are shown i Figure 6 and Table 2. The plots in the figure show estimates of *H* for packet traffic using the bin size 100 ms. The R/S method gives the highest estimate *H*=0.90 while the Variance-Time and IDC methods estimates the Hurst parameter to 0.85 and 0.84. All methods estimates *H* to more than 0.75 for both packet and byte traffic and irrespective of bin size.

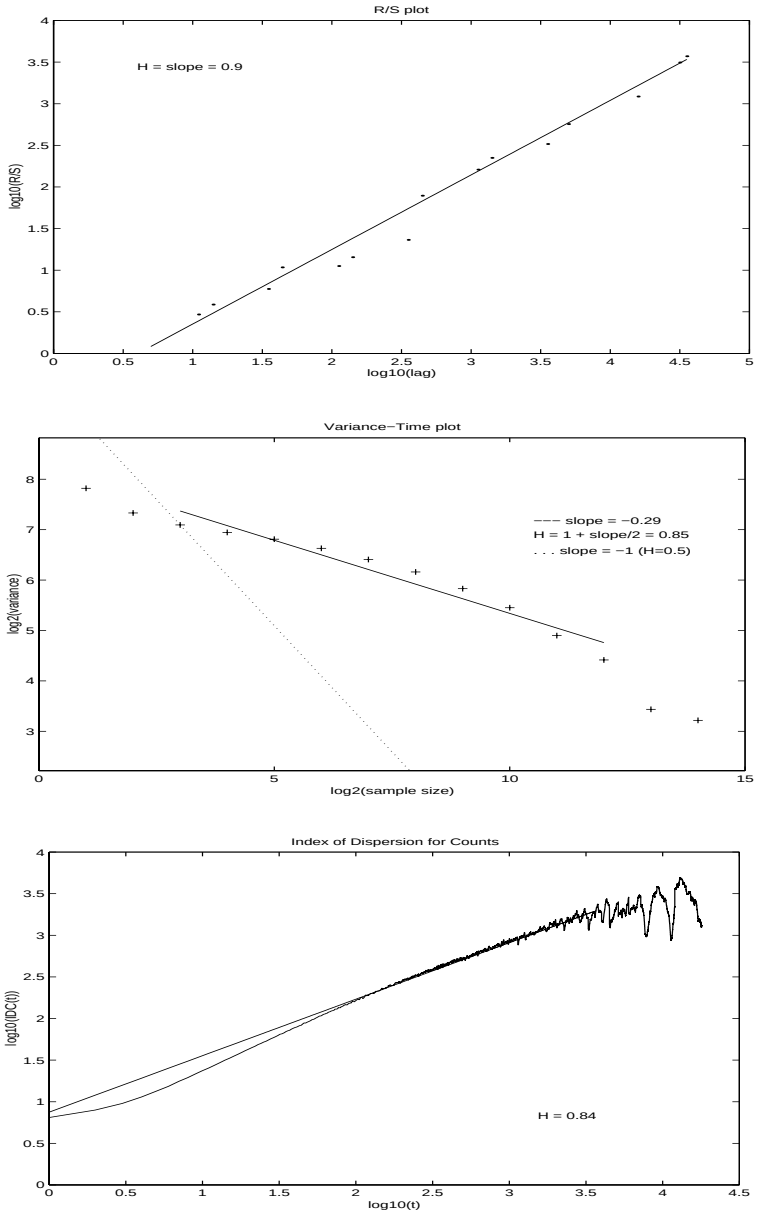

Figure 6. Estimates of the Hurst parameter H for the hour 19:42-20:42 of the Supernet trace. At the top the R/S plot followed by the Variance-Time and Index of Dispersion for Counts plots.

Bin size:	**Packets**			**Bytes**		
	H_{rs}	H_{var}	H_{idc}	H_{rs}	H_{var}	H_{idc}
0.01 s	0.82	0.88	0.84	0.78	0.83	0.79
0.1 s	0.90	0.85	0.84	0.87	0.86	0.79
1 s	0.89	0.79	0.84	0.88	0.76	0.79

Table 2. Hurst parameter estimates for the hour 19:42-20:42 of the Supernet trace.

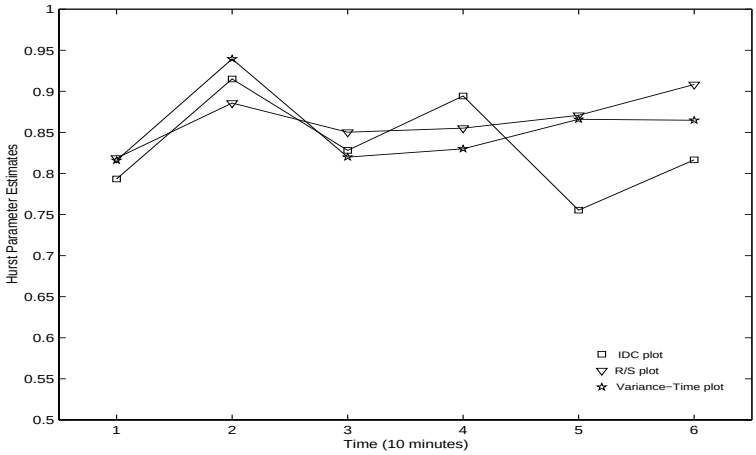

Figure 7. Hurst parameter estimates for ten minutes intervals between 19:42 and 20:42 in the Supernet trace.

The six non-overlapping ten minutes intervals between 19:42 and 20:42 was investigated and the values of the Hurst parameter is shown in Figure 7. Finally, each hour in the trace from 15:27 to 22:27 was analysed. Only the packet traffic was examined using bin size 100 ms.The results are presented in Figure 8 and show that the lowest value of H was approximately 0.75, but most estimates are above 0.85 and sometimes as high as 0.95. Thus, if the short silences between the trace files have little or no influence on the estimates then the traffic in the Supernet trace is clearly self-similar.

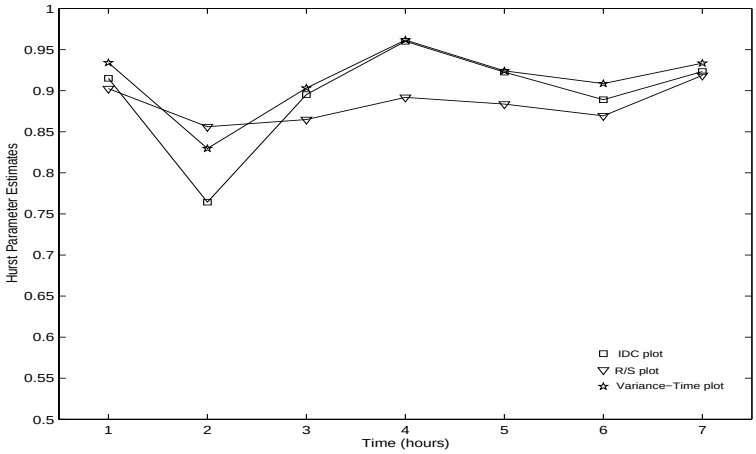

Figure 8. Hurst parameter estimates for each hour of the Supernet trace.

2.4 Self-similarity - explanations and implications

2.4.1 Why self-similarity?

The first findings of self-similarity in network traffic were met with scepticism, mainly because of the absence of physical explanations for the observed phenomena. It turned out that heavy-tailed distributions play an important part when explaining the causes of

self-similarity. These distributions have been found [32] for instance in sizes of files in a file system, inter-keystroke times when a person types, sizes of FTP bursts and in sizes and durations of bursts and idle periods in traffic between pairs of computers on an Ethernet LAN.

When trying to explain the empirically observed self-similarity, a structural modelling approach has been proposed by Leland *et al*. [18] and Willinger *et al*. [31], [33]. These structural models take into account specific features of the underlying network structure and hence provide a physical explanation for the observed fractal nature of aggregate network traffic. Aggregate Ethernet LAN traffic can be separated into individual *source-destination pairs* that represents the traffic flow between each active pair of computers. These pairs are modelled as *ON/OFF sources*. The model assumes that a source alternates between an active and an idle state. During ON-periods packets are sent at a constant rate, and during OFF-periods no packets are transmitted. The length of the ON-periods are identically distributed, and so are the OFF-periods. The length of ON- and OFF-periods are independent. Willinger *et al*. [31], [33] presents a limit theorem that states that the superposition of many such ON/OFF sources captures the self-similar nature of aggregate LAN traffic, provided that the distribution of either the ON- or OFF-periods of an individual source-destination pair are heavy-tailed with infinite variance. In [33] a data set of self-similar Ethernet LAN traffic is analysed to validate the model, and it shows that a typical individual source-destination pair exhibits an apparent ON/OFF structure and that the distribution of the ON/OFF periods satisfy the heavy-tailed property. Thus self-similar LAN traffic can be constructed by multiplexing a large number of ON/OFF sources that have heavy-tailed ON or OFF period lengths.

Traffic carried over wide-area networks such as the Internet differs from LAN traffic in some fundamental ways that makes structural modelling more complicated. WANs are generally more heterogeneous and they have to cope with delays associated with obtaining and adapting to feedback on current network conditions, which introduce additional structure to the flow of packets. Structural modelling approaches for WAN traffic have been proposed by Willinger *et al*. [33], Paxson *et al*. [21] and Feldmann *et al.* [11]. These models attempt to explain the self-similar nature of aggregate WAN traffic at the packet level in terms of the characteristics of the main applications (e.g. HTTP, FTP and Telnet). The structural models are based on a construction called the *M/G/∞ model*, where session arrivals are assumed to be Poisson, session durations are heavy-tailed and packets are generated at a constant rate for the duration of a session. These models are shown to be partly valid for todays WAN traffic, but a more flexible traffic behaviour within sessions is needed. *Multifractals*, described by Feldmann *et al*. [11], [12], is another approach to describe and understand the dynamics of WAN traffic.

Thus, it seems like the self-similarity in network traffic can be explained simply in terms of the nature of the traffic generated by individual sources.

2.4.2 Implications

Self-similarity is a way of describing the existing traffic. The network traffic was (probably) bursty and fractal-like in its behaviour even before 1994, when Leland *et al*. [18] first described it as self-similar. The fact that traffic is found to be self-similar does not

change its behaviour but it changes the knowledge about real traffic and also the way in which traffic is modelled. It has lead many [21] to abandon the Poisson-based modelling of network traffic for all but user session arrivals. Real traffic, well described as self-similar, has a “burst within burst” structure that cannot be described with the traditional Poisson-based traffic modelling.

Erramilli [10] shows, using trace-driven simulation experiments, that long-range dependence in packet traffic has measurable and practical impact on queueing behaviour. That long-range dependence is of crucial importance for buffer sizing, admission control and rate control, and if ignored typically results in too optimistic performance predictions and inadequate network resource allocation.

It should be emphasized that there is no total consensus among researchers about the importance of self-similarity and long-range dependence. The first fractal traffic models were met with scepticism. Mainly because of the absence of physical explanations, but also because it was preceded by short-lived trends of using fractals in many other areas, such as economics, hydrology and biophysics. But not many articles have been published recently that argue against the use of self-similarity in network modelling. Grossglauser and Bolot [13] does not question the evidence that network traffic exhibit properties of self-similarity and long-range dependence, but debate about their practical impact on network and application performance. They argue that processes with the same correlation structure (for instance LRD) can generate vastly different queueing behaviour. Therefore it is also important to consider other parameters for accurate performance predictions, such as the marginal distribution of the arrival process and the finite range of time scales of interest in performance evaluation.

3.0 Results of traffic measurements

Besides the question whether traffic is self-similar or not, it is also interesting to know what protocols and packet sizes are present and in what proportions. These questions are discussed in this section and the results of traffic measurements are presented. To put these results in a context, Section 3.1 gives a brief outline of the TCP/IP protocol suite. It also describes how information about protocols and packet sizes were obtained. In Section 3.2 the packet trace taken at SICS is analysed and 3.3 gives the results of the Supernet trace.

3.1 Introduction

This section gives a brief outline of the TCP/IP protocol suite. The description follows Stevens [25].

Networking protocols are normally developed in layers, each having a different responsibility. The TCP/IP protocol suite is a 4-layer system with different protocols at these layers. The *link* layer normally includes the device driver in the operating system and the corresponding network interface card in the computer. Together they handle all the hardware details of physically interfacing with the cable. The *network* layer handles the movement of packets around the network. The *transport* layer provides a flow of data between two hosts and the *application* layer handles the details of the particular application.

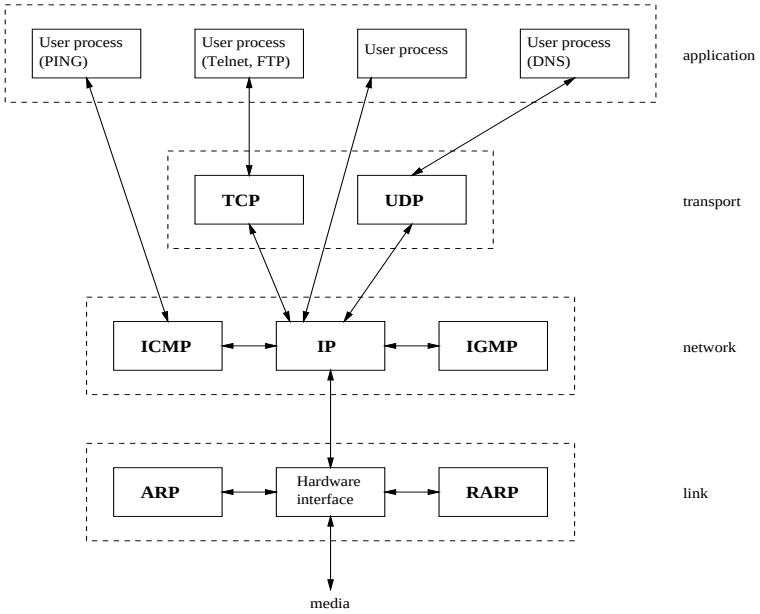

Figure 9. TCP/IP layers and protocols.

TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) are the two predominant transport layer protocols. Both use IP (Internet Protocol) as the network layer. TCP provides a connection-oriented, reliable, byte stream service to the application layer. SMTP (Simple Mail Transfer Protocol) for transferring electronic mail messages, Telnet for remote login and FTP (File Transfer Protocol) are some well-known applications that use TCP. UDP is a simpler, unreliable, transport protocol that sends and receives datagrams for applications. UDP is for instance used by DNS (Domain Name System).

IP, ICMP (Internet Control Message Protocol), and IGMP (Internet Group Management Protocol) provide the network layer in the TCP/IP protocol suite. IP is the main protocol at the network layer and it defines the IP datagram as the unit of information passed across an internet and provides the basis for connectionless, best-effort packet delivery service. It is used by both TCP and UDP. It is also possible for an application to access IP directly. ICMP is an integral part of IP that handles error and control messages. Gateways and hosts use ICMP to send reports of problems about datagrams back to the original source that sent the datagram. ICMP also includes an echo request/reply used to test whether a destination is reachable and responding. Although ICMP is used primarily by IP, it is possible for an application to also access it. For instance the Ping program uses ICMP. IGMP is used by hosts and routers that support multicasting - the sending of a UDP datagram to a group of hosts. IGMP lets all the systems on a physical network know which hosts currently belong to which multicast groups. This information is required by multicast routers, so they know which multicast datagrams to forward on which interfaces. The positioning of the ICMP and IGMP boxes in Figure 9 is not obvious. It shows them at the same layer as IP, because they really are adjuncts to IP, but ICMP and IGMP messages are encapsulated in IP datagrams.

ARP (Address Resolution Protocol) and RARP (Reverse Address Resolution Protocol) are specialized protocols used only with certain types of network interfaces, such as Ethernet and token ring, to convert between the addresses used by the IP layer and the

addresses used by the network interface. ARP provides a dynamic mapping from an IP address to the corresponding hardware address. RARP is the protocol a diskless machine uses at start-up to find its IP address. The machine broadcasts a request that contains its physical hardware address and a server responds by sending the machine its IP address. Also ARP and RARP are somewhat difficult to position in the layer hierarchy. In Figure 9 they are at the same layer as the Ethernet device driver, but they both have their own Ethernet frame type like IP datagrams.

When an application sends data, it is sent down the protocol stack, through each layer, until it is sent as a stream of bits across the network. Each layer adds information to the data by adding headers to the data that it receives. Figure 10 shows the Ethernet frame that results from an application using TCP. The result would have been similar if UDP were used but the UDP header is only 8 bytes compared to the 20 byte TCP header. When an Ethernet frame is received at the destination host it starts its way up the protocol stack and all headers are removed by the appropriate protocol.

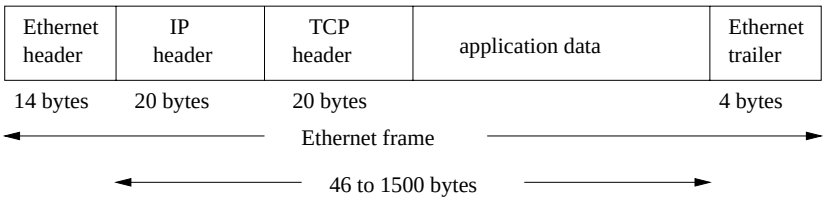

Figure 10. An Ethernet frame.

A physical property of an Ethernet frame is that the size of its data must be between 46 and 1500 bytes. The network interface sends and receives frames on behalf of IP, ARP, and RARP. To identify which protocol generated the data a 16-bit *frame type field* in the Ethernet header is used. TCP, UDP, ICMP, and IGMP all send data to IP. IP stores an 8-bit value in the *protocol field* of its header, to indicate the layer to which the data belongs. A value of 1 is for ICMP, 2 is for IGMP, 6 indicate TCP, and 17 is for UDP. Similarly, many different applications can be using TCP and UDP at the same time. Both TCP and UDP use 16-bit *port* numbers to identify applications. The source and destination port numbers are stored in the header. Servers are normally known by their *well-known* port number. For example, every Telnet server is on TCP port 23.

The unit of data that TCP sends to IP is called a *TCP segment*, the data that IP sends to the network interface is called *packet* or *IP datagram*, and the stream of bits that flows across the Ethernet is called a *frame*. But here the word *packet* will be used in a wide sense to cover all of these.

To get information about packet sizes the *tcpdump* software were used with the *-e* option to print the link-level header on each dump line. Suitable *tcpdump expressions* were chosen to select which packets should be read from the dumpfile in order to get information about protocols. For example the command:

tcpdump -e -r dumpfile port telnet

selects all Telnet packets in the file *dumpfile* and for each of them prints header information like

21:38:09.805840 [link-level info in hex] ip 60: src_host.sics.se.17209 > dst_host.se.telnet: . ack 1 win 32696 (DF)

showing the timestamp, link-level header information in hex, the packet size including the Ethernet header, source and destination IP addresses and ports etc. The UNIX commands *awk* and *egrep* were used to get timestamps and packet sizes from each dump-line.

3.2 SICS

The 24-hour packet trace taken at SICS contains header information from more than 21 million packets with a total of almost 7.6 Gigabyte (data included). Only external traffic, conversations between machines at SICS and the outside world, was captured using *tcpdump*.The utilization of the network varied a lot during the time the trace was taken. Figure 11 shows the number of packets that arrived each minute. Figure 12 shows the byte traffic.

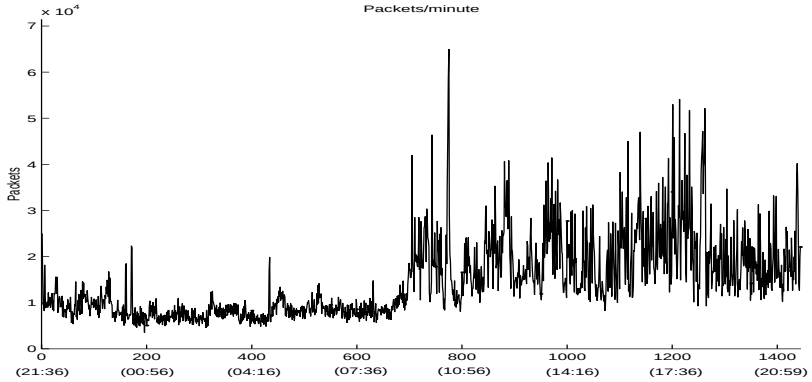

Figure 11. External packet traffic at SICS.

There was a maximum of 64971 packets in one minute at 10:31 and a minimum of 3543 packets at 00:52. These 3543 packets together contained 364234 bytes which also is the minimum number of bytes that arrived in one minute. The maximum was 39468753 bytes at 17:04.

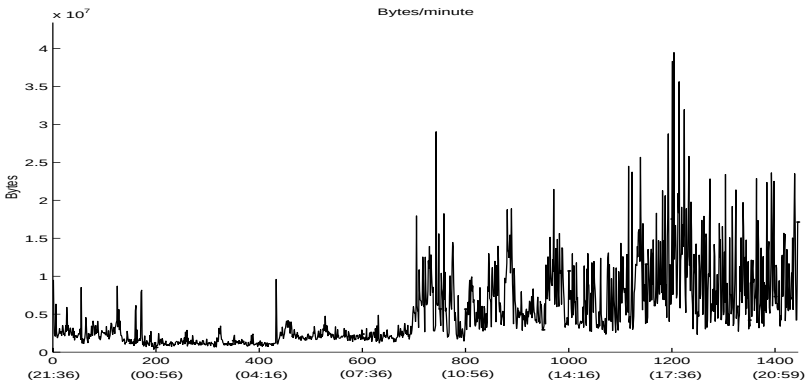

Figure 12. External byte traffic at SICS.

To be able to express the load in the more common units the number of packets and bits per one-second bin was counted. The maximum was 33.36 Mbit/sec and 4170 packets/sec. The minimum was 13.13 Kbit/sec or 9 packets/sec and the mean was 0.705 Mbit/sec and 245 packets/sec.

3.2.1 Protocols

The two tables below show the proportions in which the lower layer protocols appeared in the packet trace taken at SICS. The first shows IP versus non-IP traffic and the second shows these results in more detail.

Protocol	Packets	% of packets	Bytes	% of bytes
IP	20377055	96.17	7586281725	99.32
non-IP	810906	3.83	51565643	0.68
Total	**21187961**	**100**	**7637847368**	**100**

Table 3. IP and non-IP traffic in the SICS trace.

ICMP, IGMP and the transport protocols TCP and UDP all use IP, so the first five rows in the table below is the IP traffic. The amount of ARP, RARP and other traffic not using IP is presented in the following three rows.

Protocol	Packets	% of packets	Bytes	% of bytes
TCP	11965862	56.47	5764625001	75.47
UDP	6303931	29.75	1438035012	18.82
ICMP	156169	0.74	13209489	0.17
IGMP	154166	0.73	30929559	0.40
Other IP	1796927	8.48	339482664	4.44
ARP	589968	2.78	35398128	0.46
RARP	0	0	0	0
Other non-IP	220938	1.04	16167515	0.21
Total	**21187961**	**100**	**7637847368**	**100**

Table 4. IP and non-IP traffic in more detail.

TCP is the transport layer protocol that dominates the traffic with 56% of the packets and 75% of the bytes. Almost 30% of the packets and 19% of the bytes in the trace taken at SICS is UDP. Together TCP and UDP stands for 86% of the packets and 94% of the bytes. Note that the *Other IP* category have more than 8% of the total number of packets. These are IP packets not due to any application using TCP or UDP at the transport layer and not ICMP or IGMP packets. A closer examination shows that most of the traffic in the *Other IP* category is due to *IPv6* and *IP in IP.* As mentioned in Section 3.1 the IP header includes an 8-bit value in the protocol field which identify the next level protocol. 83% of the *Other IP* packets (comprising 57% of the bytes) have a protocol field value of 41, which is the IPv6 protocol. So, approximately 7% of the total number of packets (and 2.5% of the bytes) is IPv6 packets sent in ordinary IPv4 packets. 16% of the *Other IP* packets and 41% of the bytes is due to IP in IP. That is IP packets encapsulated (carried as payload) within other IP packets. Encapsulation is a means to alter the normal IP routing by delivering packets to an intermediate destination that would otherwise not be selected based on the destination address in the origi-

nal IP header. When the encapsulated packet arrives at this intermediate destination it is decapsulated, yielding the original IP packet which is sent to the destination indicated by the original destination address. This use of encapsulation and decapsulation of a packet is called *tunneling*. A common application is multicasting.

The composition of the IP traffic during the time the trace was taken can be seen in the figures below. Figure 13 shows composition of packets and Figure 14 composition of byte volume by IP protocols. The *Other IP* category here includes ICMP and IGMP.

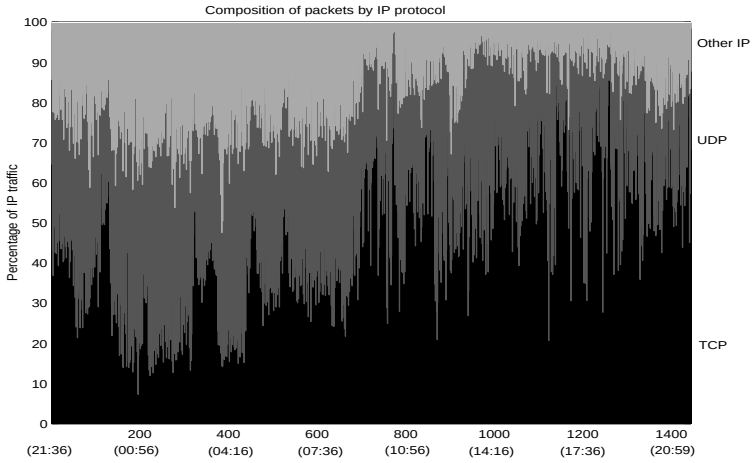

Figure 13. Compostion of packets in the SICS trace by IP protocol.

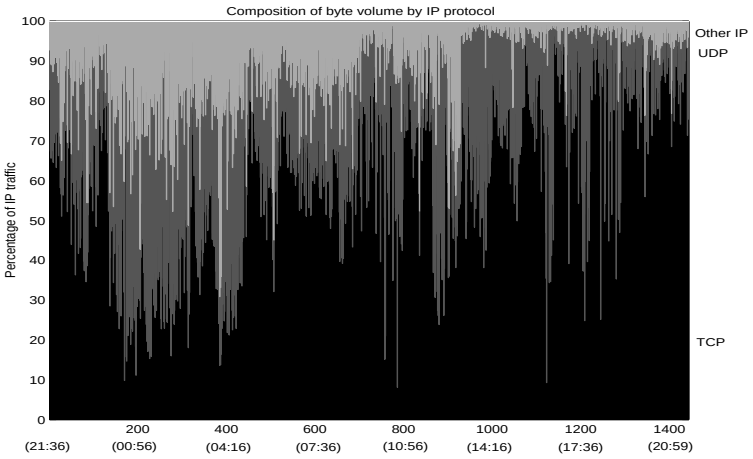

Figure 14. Composition of byte volume in SICS trace by IP protocol.

The TCP traffic comprises a large part of the traffic but, compared to recent measurements on Internet backbone [6],[29] where TCP averages about 95% of the bytes and 90% of the packets, the external traffic at SICS also includes a lot of UDP and other traffic.

To get an idea of which application protocols are most used, the source and destination-ports of all TCP and UDP packets were taken out of the trace using *awk* and the number of occurrences of each portnumber were counted. For the most common protocols more detailed information, such as the exact number of packets and bytes, were

then obtained using *tcpdump* with appropriate boolean expressions. Table 5 shows these results but also information about some well known protocols, like Telnet for instance, that don't comprise a very large part of the traffic.

Protocol	Packets	% of packets	Bytes	% of bytes
NNTP	4745322	25.97 (22.40)	2805319120	38.95 (36.73)
DNS	3322824	18.19 (15.68)	451128865	6.26 (5.91)
kshell	2361908	12.93 (11.15)	1089165179	15.12 (14.26)
HTTP	1910608	10.46 (9.02)	1049716285	14.57 (13.74)
NetBIOS	1058226	5.79 (4.99)	185667131	2.58 (2.43)
NFS	957975	5.25 (4.52)	417610717	5.80 (5.47)
SMTP	568871	3.11 (2.68)	120384357	1.67 (1.58)
FTP	379186	2.08 (1.79)	169015460	2.35 (2.21)
Telnet	128610	0.70 (0.61)	12619392	0.18 (0.17)
NTP	27538	0.15 (0.13)	2529840	0.035 (0.03)
Other	2809279	15.38 (13.26)	899611097	12.49 (11.78)
Total	**18270347**	**100.01 (86.2)**	**7202767443**	**100.00 (94.3)**

Table 5. Application protocols using TCP or UDP.

The third column shows the percentage of all TCP and UDP packets and the figures put in parenthesis is percentage of the total number of packets. The total number of TCP and UDP packets are 18269793 with a total of 7202660013 bytes. There are some traffic (554 packets actually) between the NetBIOS ports and the domain or nfs ports, so some packets are counted for twice. The *Other* category have more than 15% of the TCP and UDP packets spread among a lot of different ports, non of which represents more than 3% of the packets.

The most common application protocol is the NNTP (Network News Transfer Protocol). More than 22% of all packets are due to NNTP, which is an application protocol that uses TCP to distribute news articles between cooperating hosts.

The second most common application is DNS (Domain Name System), the on-line distributed database system used to map human-readable machine names into IP addresses. More than 15% of the total number of packets in the trace is DNS and almost 6% of the bytes. DNS uses both TCP and UDP. A DNS client (called resolver) is normally part of a client application, for example, a Telnet client, an FTP client, or a WWW browser. The resolver sends a single UDP datagram to a DNS server requesting the IP address associated with a domain name. The reply is normally a single UDP datagram from the server, but if the reply exceeds 512 bytes only the first 512 bytes are returned along with a flag indicating that more information is available. The client then resends the query using TCP and the server returns the entire reply using TCP [26].

The TCP port 544, kshell (Kerberos remote shell), is the third most common port in the SICS trace. It is used for secure remote login and file transfer.

HTTP (Hypertext Transfer Protocol) is the basis for the World Wide Web (WWW). HTTP messages are transported by TCP connections between clients (web browsers)

and servers. HTTP dominates the information exchange on the Internet. Measurements on Internet backbone [6], [29], shows that HTTP comprises 75% of the overall bytes and up to 70% of the overall packets. But in the trace taken at SICS less than 10% of all packets are HTTP.

NetBIOS (Network Basic Input Output System) in the table above refers to all packets with source or destinationports netbios-ns 137 (nameservice), netbios-dgm 138 (datagram service) or netbios-ssn 139 (session service). Put together these packets make up about 5% of the traffic. NetBIOS was originally developed by IBM as an Application Program Interface (API) for IBM PC programs to access LAN facilities. In the TCP/IP internet, NetBIOS refers to a set of guidelines that describes how to map NetBIOS operations into equivalent TCP/IP operations.

Another 5% of the traffic is due to NFS (Network File System), a protocol that uses IP to allow a set of cooperating computers to access each other's filesystems as if they were local. SMTP, FTP and Telnet all have low byte and packet percentage. Together they make up 5-6% of the packet traffic. FTP includes traffic using both the ftp and the ftp-data ports. NTP (Network Time Protocol) is a protocol used for maintaining the clocks for a group of systems on a LAN or WAN to within millisecond accuracy. NTP constitutes only 0.13% of the packets.

3.2.2 Packet sizes

The smallest packets in the trace were 60 bytes and the largest 1514 bytes. Figure 15 below shows the relative frequency of packet sizes including the 14 byte Ethernet header.

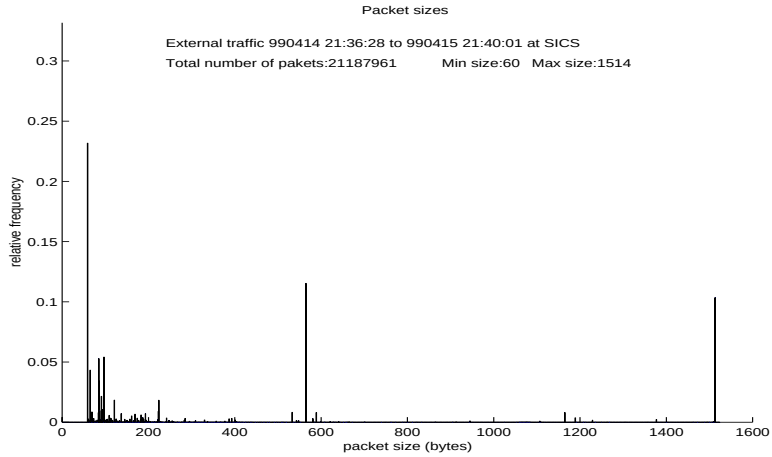

Figure 15. Relative frequency of packet sizes in the SICS trace.

The ten most common packet sizes are 60, 566, 1514, 98, 86, 66, 87, 92, 122 and 225 bytes. There are peaks at sizes 60, 566 and 1514 bytes. The small packets, 60 bytes in length, include TCP acknowledgement packets, TCP control packets such as SYN, FIN and RST packets, and Telnet packets carrying single characters (keystrokes of a telnet session). A packet containing a TCP acknowledgement does not include any data except for the TCP and IP headers (20 +20 = 40 bytes) so the packet size of 60 bytes is due to the fact that the minimum data portion of an Ethernet packet is 46 bytes and a pad field is used to fill out the frame to the minimum size. The minimum 46 bytes of data plus the 14 bytes Ethernet header make up the 60 bytes packet size.

Many TCP implementations use 512 bytes as the default Maximum Segment Size (MSS) for non-local IP destinations, yielding a 512+20+20+14 = 566 byte packet size. Each network has a Maximum Transfer Unit (MTU) - the largest amount of data that can be transferred across a given physical network. A MTU size of 1500 is characteristic of Ethernet attached hosts.

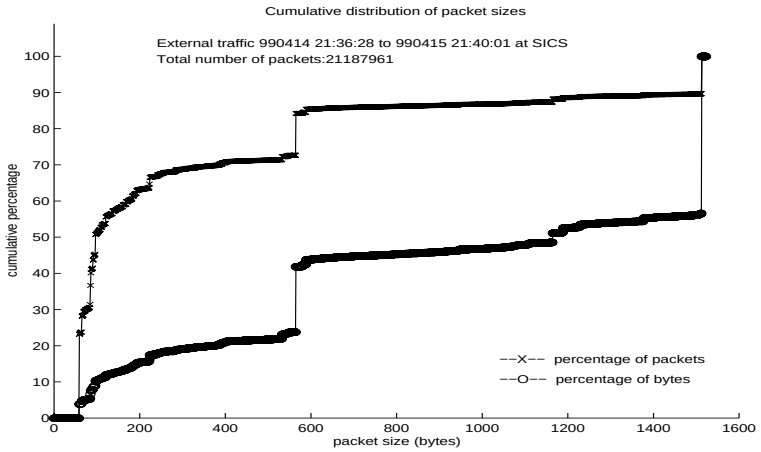

Figure 16. Cumulative distribution of packet sizes and of bytes by the size of the packet carrying them.

Figure 16 shows the cumulative distribution of packet sizes, and of bytes by the size of packets carrying them. Most of the packets are small but most of the bytes are transferred in large packets. Half of the packets are less than 100 bytes. 73% of the packets are smaller than the common TCP maximum segment size (566 bytes with headers included) but more than 75% of the bytes are carried in packets of size equal to or more than 566 bytes. Less than 11% of the packets have the maximum size, 1514 bytes, but almost 45% of the bytes are transferred in packets of this size.

3.3 Supernet

The packet trace taken at Supernet contains header information from exactly 15 million packets. These packets make up a total of more than 12 Gigabytes. The number of packets and bytes that arrived in each one-minute bin was counted. Figure 17 shows how the packet traffic varies during the time the trace was taken. Figure 18 shows the byte traffic.

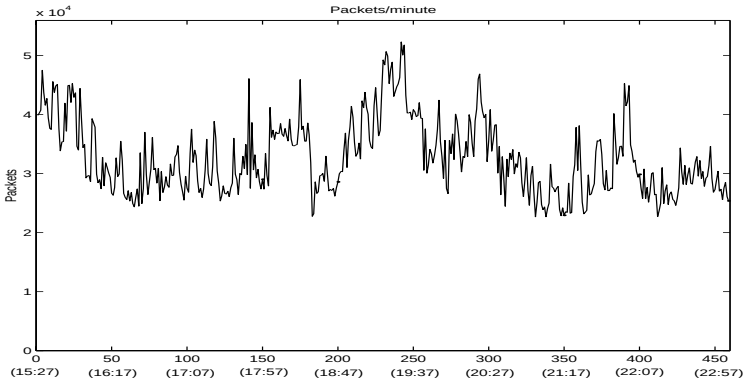

Figure 17. Packet traffic in Supernet trace.

There are minimums of 22664 packets/minute at 20:58 and 20595624 bytes/minute at 22:19. The maximum number of packets in one minute is 52264 at 19:29 and the maximum number of bytes/minute is 44223918 at 18:22.

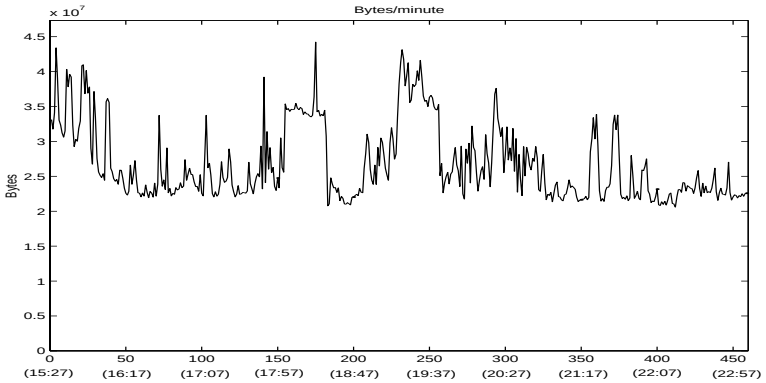

Figure 18. Byte traffic in Supernet trace.

To express the load in the more common units the number of packets and bits per one-second bin was counted. The maximum was 9.68 Mbit/sec and 1560 packets/sec. The minimum was 1.28 Mbit/sec or 175 packets/sec and the mean was 3.6 Mbit/sec and 544 packets/sec.

3.3.1 Protocols

The presentation of the results is done in the same way as for the SICS trace. The two tables below show the proportions in which the lower layer protocols appeared in the packet trace.

Protocol	Packets	% of packets	Bytes	% of bytes
IP	14612623	97.42	12294871647	99.68
non-IP	387377	2.58	39557446	0.32
Total	**15000000**	**100**	**12334429093**	**100**

Table 6. IP and non-IP traffic in Supernet trace.

Table 6 shows IP versus non-IP traffic. Table 7 shows these results in more detail with the IP traffic divided up in TCP, UDP, ICMP and IGMP traffic.

Protocol	Packets	%of packets	Bytes	% of bytes
TCP	3839247	25.59	2060668280	16.71
UDP	10616430	70.78	10221812908	82.87
ICMP	119084	0.79	9099175	0.074
IGMP	3876	0.026	232560	0.0019
Other IP	33986	0.23	3058724	0.025
ARP	68241	0.45	4094460	0.033
RARP	485	0.0032	29100	0.00024
Other non-IP	318651	2.12	35433886	0.29
Total	**15000000**	**100**	**12334429093**	**100**

Table 7. IP and non-IP traffic in more detail.

There is a very large amount of UDP traffic in this trace. More than 70% of the packets and 80% of the bytes are UDP which is a big difference compared to the SICS trace. UDP and TCP together comprises more than 96% of the packets and 99% of the bytes. The *Other IP* category have less percentage of the traffic here than at SICS. The figures below shows the composition of the IP traffic during the time the trace was taken. Figure 19 shows the composition of packets and Figure 20 the composition of byte volume by IP protocols

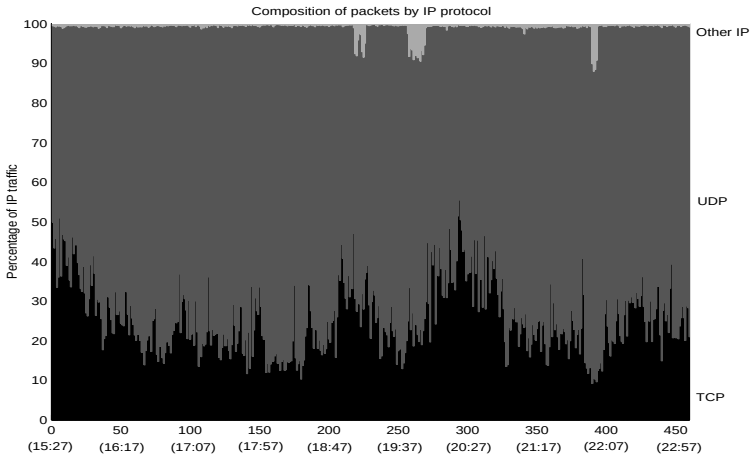

Figure 19. Composition of packets in Supernet trace by IP protocol.

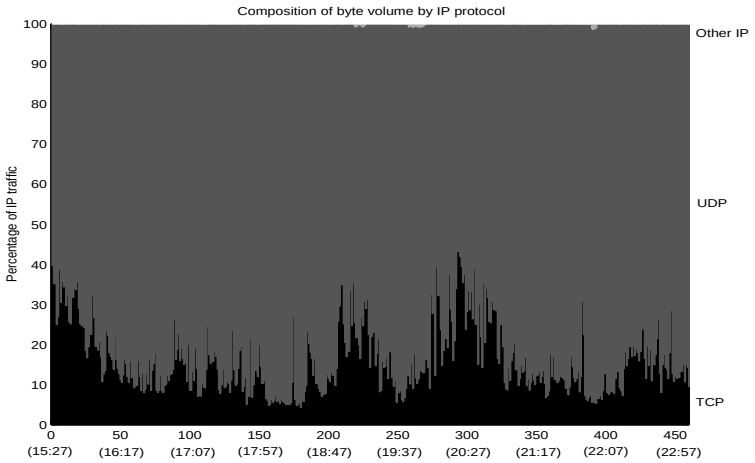

Figure 20. Composition of byte volume in Supernet trace by IP protocol.

The Other IP category here also includes ICMP and IGMP. Notice in Table 7 above that there is a small amount of RARP traffic in this trace which didn't occur in the SICS trace.

The trace is known to include two TV channels and one radio channel sent as unicast. There should also be a lot of game playing (Quake). Without this knowledge, it would not be easy to explain what applications are used on the Supernet by analysing the packet trace. The well known application protocols make up only a minor part of the traffic. Table 8 shows the most common ports and the number of packets and bytes in the trace sent to or from these ports.

	Packets	**% of packets**	**Bytes**	**% of bytes**
port 2048	4262364	29.49 (28.42)	4599203709	37.45 (37.29)
port 1032	1448021	10.02 (9.65)	1417213223	11.54 (11.49)
HTTP	957682	6.62 (6.38)	484476726	3.94 (3.93)
port 7070	617031	4.27 (4.11)	322146669	2.62 (2.61)
NetBIOS	506173	3.50 (3.37)	83768506	0.68 (0.68)
port 5501	488216	3.38 (3.25)	451899542	3.68 (3.66)
port 1042	472751	3.27 (3.15)	132064762	1.08 (1.07)
port 1090	334116	2.31 (2.27)	360962491	2.94 (2.93)
port 1267	328932	2.28 (2.19)	355555718	2.89 (2.88)
port 9000	307638	2.13 (2.05)	22765116	0.19 (0.18)
FTP	247729	1.71 (1.65)	189000564	1.54 (1.53)
Other	4498253	31.12 (29.99)	3865965233	31.48 (31.34)
Total	**14467833**	**100.1 (96.5)**	**12282481188**	**100.0 (99.6)**

Table 8. Application protocols using TCP or UDP.

The total number of TCP and UDP packets were 14455677. There is some traffic (12156 packets) between the ports in the table above. These packets are counted for twice. The *other* category have 31% of both packets and bytes spread among a wide range of TCP and UDP port numbers. None of these ports represents more than 1.7% of the total number of packets.

The traffic with source or destination port 2048 dominates the traffic with almost 30% of the packets and 37% of the bytes. 99.6% of this traffic is UDP traffic between the same two hosts with the other port being 8003. The traffic is very smooth with approximately 9250 packets/minute during the interval the trace was taken.

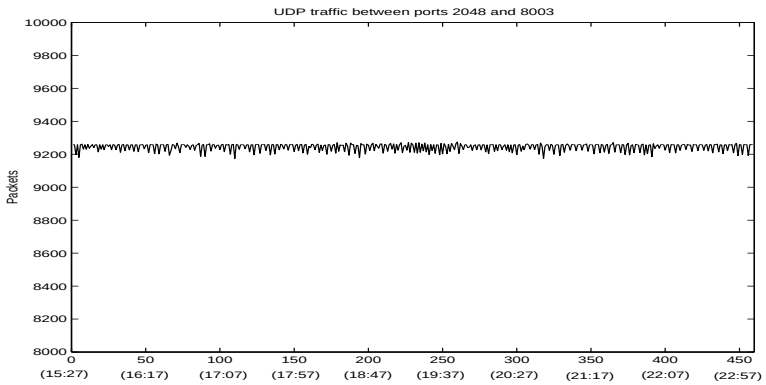

Figure 21. UDP traffic in Supernet trace between the same two machines using ports 2048 and 8003.

Almost all (99.29%) of the port 1032 traffic is UDP traffic between two hosts with the other port being 8002. This traffic is as smooth as the port 2048 traffic.

99.5% of the port 7070 (ARCP) traffic is TCP traffic.

NetBIOS in Table 8 is all the traffic to the three NetBIOS ports(137-139) put together.

All of the port 5501(fcp-addr-srvr2) traffic is TCP. This port is used on three occasions in the trace. 96% of this traffic is between the same two hosts. None of the traffic coincide with traffic to TCP ports 7070, 1042, http or the ftp ports.

Almost all (99.7%) of the traffic with source or destination port 1042 is TCP traffic. 99.8% of this traffic is between the same two hosts with the other port being 40094.

99.8% of the traffic to or from the ports 1090, 1267 (both unassigned) and 9000 (CSlistener) is UDP.

There are four (!) NFS packets in the trace. Note that the tcpdump software distinguish between NFS and traffic to port 2049. The tcpdump expression *port 2049* does not give the same result as *port nfs*.

The trace contains 25217 DNS packets using either TCP or UDP with a total of 2632572 bytes. That is 0.16% of the total number of packets and 0.00021% of the bytes.

There is no NTP packets in the trace.

UDP traffic dominates the above table of the most common application traffic. The table below shows only TCP traffic including the well-known SMTP, Telnet and NNTP traffic.

	Packets	% of packets	Bytes	% of bytes
HTTP	957682	25.59 (6.38)	484476726	23.51 (3.93)
port 7070	614206	16.00 (4.09)	320592324	15.58 (2.60)
port 5501	488216	12.72 (3.25)	451899542	21.93 (3.66)
port 1042	472691	12.31 (3.15)	132059486	6.41 (1.07)
FTP	247729	6.45 (1.65)	189000564	9.17 (1.53)
SMTP	45079	1.17 (0.30)	16086871	0.78 (0.13)
Telnet	6627	0.17 (0.0004)	469120	0.023 (4e-5)
NNTP	8	2.1e-4 (5e-7)	546	2.6e-5 (4e-8)
Other	1008147	26.26 (6.72)	466512715	22.64 (3.78)
Total	**3840385**	**100.7 (25.5)**	**2061097894**	**100.0 (16.7)**

Table 9. Application protocols using TCP.

The total number of TCP packets were 3839247. There is some traffic (1138 packets) between the ports in the table above. These packets are counted for twice. The *other* category have 26% of the packets and 23% of the bytes spread among a wide range of TCP ports. None of these ports represents more than 4% of the total number of TCP packets. Notice that the trace only contains eight NNTP packets. In the SICS trace NNTP was dominating the traffic with 22% of all packets.

3.3.2 Packet sizes

The smallest packets were 43 bytes and the largest 1514 bytes.

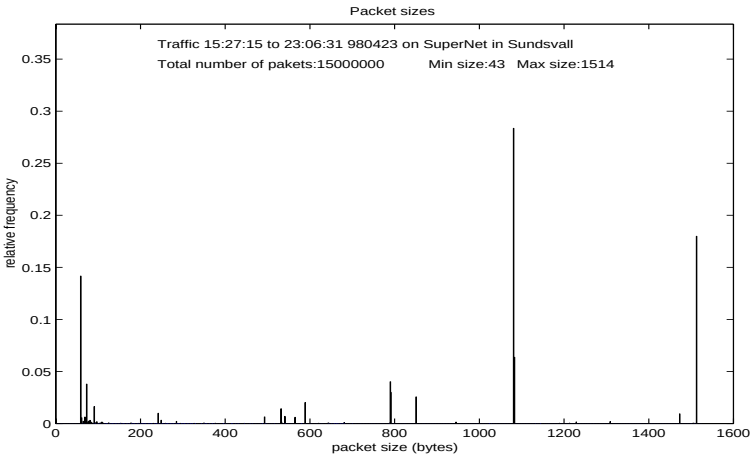

Figure 22. Relative frequency of packet sizes in the Supernet trace.

The ten most common packet sizes are 1082, 1514, 60, 1084, 791, 74, 792, 852, 590 and 92 bytes including the 14-byte Ethernet header. Almost all of the packets (99.8%) of size 1082 bytes is UDP packets between the same two hosts using ports 2048 and 8003. The other two peaks are the same as described for the SICS trace. 1500 bytes is

the maximum packet size for Ethernet attached hosts and small packets are padded to the minimum size 46 bytes. Add the 14 bytes Ethernet header and you get the common packet sizes 1514 and 60 bytes. With this in mind it somewhat peculiar that the minimum packet size found in the trace is 43 bytes. There are 9155 packets in the trace that are smaller than 60 bytes. A more detailed analysis shows that small UDP packets are not padded to the minimum size. In the trace taken at SICS small UDP packets are padded and in both traces are small TCP packets padded to the minimum 46 bytes.

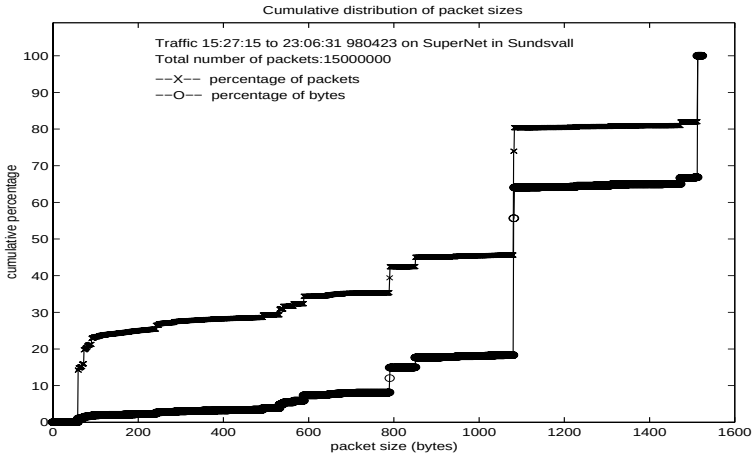

Figure 23. Cumulative distribution of packet sizes and of bytes by the size of the packets carrying them.

Figure 23 shows the cumulative distribution of packet sizes, and of bytes by the size of packets carrying them. Most of the packets are large. 55% of the packets are 1082 bytes or more. More than 80% of the bytes are transferred in packets larger than or equal to 1082 bytes in size. This is very different from the packet size distribution at SICS and different from measurements made at Internet backbones [29], where most of the packets are small. The UDP traffic between the same two hosts generates almost all of the packets of size 1082 which have a great influence on the distribution in Figure 23. Figure 24 shows the cumulative distribution of packet sizes with this traffic excluded. There are still a lot of large packets and only slightly more than 30% of the packets are smaller than 100 bytes.

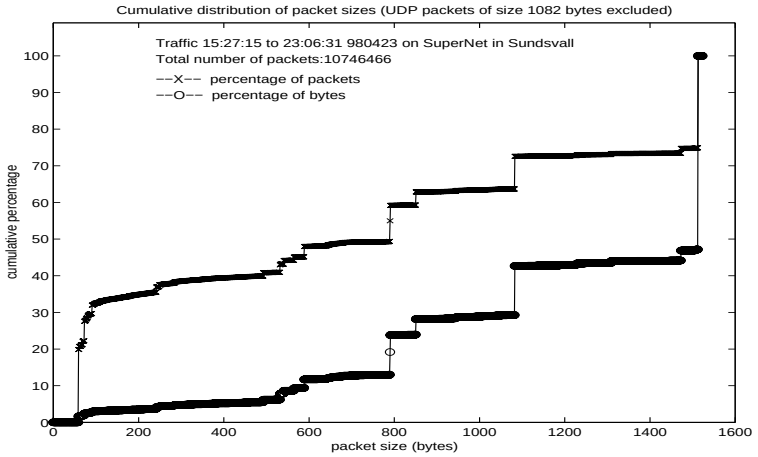

Figure 24. Cumulative distribution of packet sizes with UDP packets of size 1082 bytes excluded.

4.0 Modelling HTTP traffic

HTTP (Hypertext Transfer Protocol) is the basis for the World Wide Web (WWW). Recent measurements on Internet backbone [6],[29] show that HTTP comprises approximately 70-75% of the total traffic. Here HTTP traffic captured at SICS is analysed to obtain traffic characteristics of a single client. Empirical probability distributions are derived describing session lengths, time between user clicks and the amount of data that is transferred due to a single user click. These probability distributions make up a simple model of WWW-sessions that can be used to generate traffic. The definition of *session* varies between different authors and different contexts. Here a *session* is defined to be an interval in which a user creates WWW-traffic without being silent for more than a certain time. How the session boundaries were determined in detail is described in section 4.2.

The model is not dealing with the number, size and interarrival times of TCP packets since these quantities are governed by the TCP flow control and congestion control algorithms. The timing of a connection's packets as recorded in a trace reflects the conditions in the network at the time the connection occurred [22]. Due to this adaptation to the network done by TCP, a trace of a connection's packets cannot be reused in another context, because the connection would not have behaved the same way in the new context.

The *tcpdump* software [15] was used to capture the HTTP traffic. To extract useful information from the trace, about the client's behaviour, knowledge about the HTTP protocols and their use of TCP is needed. Section 4.1 describes the protocols HTTP/1.0 and HTTP/1.1 and how they are used by web browsers. Section 4.2 describes in more detail how information was extracted from the packet trace and in 4.3 the results are presented.

4.1 The HTTP protocols

The application-level protocol HTTP exists and is used in more than one version, but there is yet no formal standard that everybody follows. HTTP/1.0 evolved from the original 0.9 version of HTTP (which is still in rare use). The HTTP Working Group (HTTP-WG) of the Internet Engineering Task Force (IETF) produced the document RFC 1945 [23] that described the common usage of HTTP/1.0, but did not attempt to create a formal standard out of the many variant implementations. Instead, over a period of roughly four years, the working group has developed an improved protocol, known as HTTP/1.1. Because the HTTP/1.1 effort took over four years, and generated numerous draft documents, several pseudo-HTTP/1.1 versions were created and used by many implementors. When this is written HTTP/1.1, as it is described in RFC 2616 [24], has recently (7 july 1999) been approved by the Internet Engineering Steering Group (IESG) of the IETF, as a IETF Draft Standard. A Draft Standard is the second of the three step IETF standardization process, and is considered to be close to a final specification.

4.1.1 HTTP/1.0

HTTP is a simple protocol. The client establishes a TCP connection to the server, issues a request, and reads back the server's response. The server indicates the end of its response by closing the connection.

Three types of HTTP/1.0 requests are supported: GET, HEAD and POST. The format of a request is:

request-line
headers
<blank line>
body (only if POST request)

The format of an *request-line* is: *request request-URI HTTP-version.*

Each web page has a URI (Uniform Resource Identifier) that effectively serves as the page's worldwide name. The GET request returns whatever information is identified by the request-URI. The HEAD request is similar to the GET request, but only the servers header information is returned, not the actual contents of the specific document. This request is used to test a hypertext link for validity, accessibility, and recent modification. The POST request is used for posting electronic mail, news, or sending forms that can be filled in by an interactive user. This is the only request that sends a body with the request. In a sample of 500,000 client requests 99.68% were GET, 0.25% were HEAD, and 0.07% were POST [26]. This of course depends somewhat on the applications available at the server.

The format of an HTTP/1.0 response is

status-line
headers
<blank-line>
body

The *status-line* begins with the HTTP-version, followed by a 3-digit numeric response code, followed by a human-readable response phrase. For instance code 200 (OK), 304 (not modified) or 400 (bad request). With HTTP/1.0 both requests and responses can contain a variable number of header fields. For more information Stevens [26] lists seventeen different headers and also in detail explains the different response codes and phrases used by HTTP/1.0. Following the last response header, the server sends a blank line followed immediately by the data, for instance an HTML document, an image or a PostScript file.

As mentioned above, HTTP uses TCP as its transport protocol. When a browser using HTTP /1.0 is used to fetch web pages, a new TCP connection is set up for each document requested. Web pages often have many embedded images, and each image is retrieved via a separate HTTP request. Thus, to retrieve a web page with five images, six different TCP connections are required. The first TCP connection transfers an HTTP GET request to receive the HTML document that refers to the five images. A very simple browser would, when the HTML document is received, open one new TCP connection to get the first image. After sending the response the connection is closed

by the server and another connection is opened to get the second image and so on. The use of a new TCP connection for each image serializes the display of the entire page. Netscape introduced the use of parallel TCP connections to compensate for this serialization. When the HTML document is received four TCP connections are opened in parallel for the first four images. Stevens [26] show that for Netscape 1.1N simultaneous connections decreases the transaction time for the user.

4.1.2 HTTP/1.1

HTTP/1.1, as it is described in RFC 2616 [24], differs from HTTP/1.0 in numerous ways, both large and small. For instance there are 24 new status codes in HTTP/1.1, several new header fields and besides the GET, HEAD and POST requests used in HTTP/1.0 there are 5 other request methods specified. The OPTION method represents a request for information about the communication options available on the request/response chain identified by the Request-URI. This method allows a client to determine the options and requirements associated with a resource, or the capabilities of a server, without initiating a resource retrieval. Here a resource is defined to be a network data object or service that can be identified by a URI. The PUT method is somewhat similar to the POST method and requests that the information transferred as payload of the request be stored under the supplied Request-URI. The DELETE method requests that the server delete the resource identified by the Request-URI. The TRACE method allows the client to see what is being received at the other end of the request chain and use that data for testing and diagnostic information. The specification also reserves the name CONNECT for use with a proxy that can dynamically switch to being a tunnel.

The PUT and DELETE requests also exists in some versions of HTTP/1.0. RFC 1945 [23] describes them in an appendix as protocol elements used by some existing implementations, but not consistently and correctly across most HTTP/1.0 applications.

Krishnamurthy *et al*. [17] describes the major differences between HTTP/1.0 and HTTP/1.1 and the rationale behind them. They divide the protocol changes into nine major areas. Of most interest here is the network connection management. The problem in HTTP/1.0 that a new TCP connection is required for each document is resolved by the use of *persistent connections* and the *pipelining* of requests on a persistent connection. Persistent connections means that the client and server keep a TCP connection open instead of the server closing the connection after sending the response. The same connection can be used to fetch several images and is kept open even if the user clicks to another web page as long as the page is located on the same server. Pipelining means that a client can send an arbitrarily large number of requests over a TCP connection before receiving any of the responses. Each request must still be sent in one contiguous message and a server must send responses (on a given connection) in the order that it received the requests.

HTTP/1.0, in its documented form, made no provision for persistent connections but some implementations use a Keep-Alive header to request that a connection persist.

4.1.3 An example

To get an idea of how HTTP and TCP actually are used when a client is surfing the net, two common web browsers were tested, and the TCP traffic that arise when web pages

were fetched was captured using *tcpdump*. Some information about the HTTP requests and responses, such as request method, URI, version and response codes, can be obtained from the *tcpdump* packet trace by looking at the raw packets in plain text.

The browsers Netscape 4.6 and Microsoft Internet Explorer 4.0 were used to visit the homepage of Uppsala University (http://www.uu.se) and following the link *English* to get a presentation in english (http://info.uu.se/presengl.nsf) and then the link *Nobel prizes* to see a presentation of the eight Nobel laureates that have been connected with the University (http://info.uu.se/presengl.nsf/0/1....*)*. The homepage includes eleven gif images, the english presentation page includes 5 and the page about nobel prizes includes 9 different images. The images are all located at the same addresses as the pages, thus all images on the homepage have address http://www.uu.se/images/... . The web pages are not located on the same server. The tcpdump trace shows that the homepage is located on a server named *columba* and the other two on the server *info*. This is of importance for the use of persistent connections, which only can be held open if the pages are located on the same server.

Figure 25 shows a time line for the TCP connections. Each arrow represents a TCP connection and for each connection all HTTP client requests transferred are shown. To make it easier to compare the behaviour of the two browsers, the times between user clicks have been converted to the same values for both browsers. The purpose of this example is not to explain the complete behaviour of these browsers or to evaluate which browser is the best, but to describe how HTTP and TCP is used.

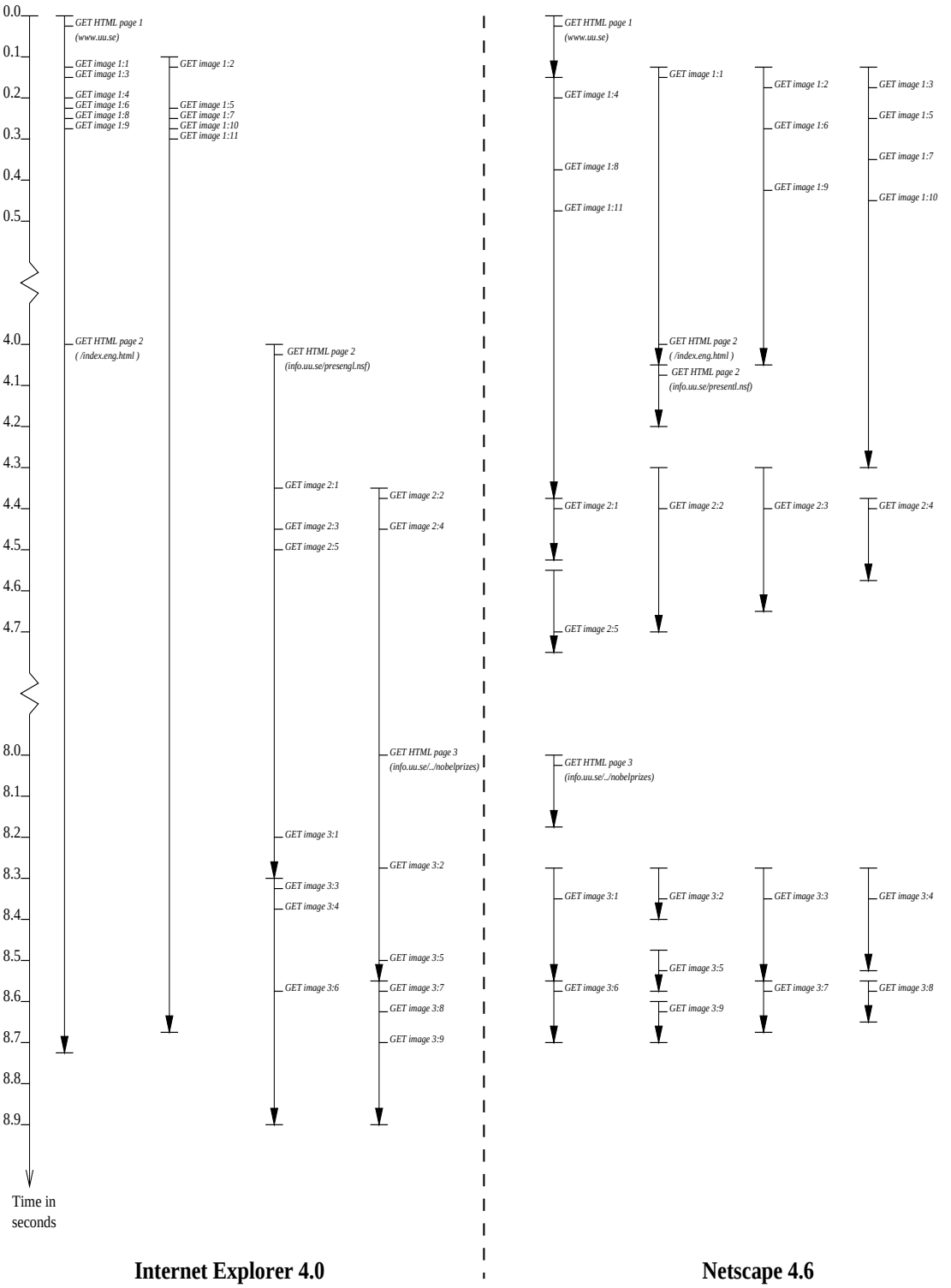

Figure 25. Comparison between how the browsers Internet Explorer 4.0 and Netscape 4.6 use TCP and HTTP when fetching three web pages from Uppsala University.

In Figure 25 it shows that two requests are made to get the second web page with a presentation in english. The first one, GET /index.eng.html, is an immediate consequence of the user clicking on the link *English* to get to this page. However the response from the server to this request is response code 301 (Moved Permanently).

This means [24] that the requested resource has been assigned a new permanent URI and any future reference to this resource should use the returned URI. The new permanent URI is given by the Location field in the response. When the client receives this response a second request for the page is made, now using the new URI (http://info.uu.se/presengl.nsf).

In this example Internet Explorer uses six different TCP connections to get the three web pages whereas Netscape uses twenty one connections. Both browsers (sometimes) holds a connection to get more than one image, but only Internet Explorer holds a connection to get images from different web pages. The TCP connection used to get the second and fourth image to the english presentation web page (the second page) is held open and is used, after the user click, to get the third web page.

4.2 Methodology

4.2.1 Prior work

Three approaches have been widely used in investigating the WWW traffic: server logs, client logs and packet traces. For different reasons most web servers keep logs of the requests they have served and these logs can be processed and used to characterize the traffic at the server. But server logs cannot easily be used to describe the client side since a client usually access many different web servers. To capture the user accesses between multiple servers client logs can be used. This approach was used by Catledge [5], Cunha [7], Crovella [8] and Barford [1] when investigating various characteristics of web accesses. This approach requires that browsers can log their requests, that the source code for the browser is available so that logging can be added, or some other way to log the clients behaviour. In [5], [7] and [8] instrumented versions of the Mosaic web browser were used, and in [1] HTTP proxies were used to track all documents referenced by unmodified Netscape Navigator clients. The third approach of gathering data, and the method used here, is to analyse packet traces taken from a subnet carrying HTTP traffic. With knowledge about the HTTP and TCP protocols the packet trace can be analysed and used to model the web traffic. This method is used by Stevens [26] to analyse the traffic arriving at a server, and by Mah [19] and Vicari [30] to model the client side of the HTTP traffic.

4.2.2 The packet trace

To analyse the HTTP traffic a 24 hour packet trace was used. This trace, taken at SICS, was described in detail in Section 2 and 3. The trace was taken by using *tcpdump* with the *-w* option so that the raw packets were written to file. In this way the trace could be used many times as input to *tcpdump* to filter out certain aspects of the traffic using different *tcpdump* options. In this case only the HTTP traffic was of interest. HTTP communication usually takes place over TCP/IP connections. The default port is TCP 80, but other ports can be used. In an examination of WWW pages Woldruff *et al*. [34] analysed 2.6 million HTML documents collected by the Inktomi Web crawler, and show that approximately 94% of the documents investigated were accessed using the standard HTTP port 80. By using the *tcpdump* option to gather all packets to or from this port a vast majority of the HTTP traffic was captured. Only traffic where users at SICS were clients was captured, not the HTTP traffic that arise from people outside

visiting the SICS web pages. The trace contains 1339209 packets transferred between TCP port 80 on web servers and 106 different clients at SICS.

4.2.3 Sessions

The definition of a *session* varies between different authors and different contexts. For instance Vicari [30] defines a WWW-session as the period starting at the time a user launches his WWW-browser and ending when the user quits the browser. He also defines a *sub-session* to be the interval in which a user creates WWW-traffic without being silent for more than certain time. Barford *et al.* [1] and Crovella *et al.* [9] defines a *session* as a single execution of the browser. Barford and Crovella [3] defines a *browsing session* as the period during which Web objects are transferred with intervening idle periods.

The notion of a *session* is supposed to cover the time interval when a user is active and uses the browser to fetch and read web pages (or listen to them, watch them or whatever the applications are). A *session* starts when the first web page is fetched and ends when the user stops surfing the net to get some work done. This is vague and difficult to exactly define in terms of packets sent and received. Since it can't be determined from a packet trace when a user starts and quits the browser and since users often leave the browser running for extended periods of time without interacting with it, determining session boundaries artificially is necessary. Here a *session* is defined to be something close to what Vicari calls a sub-session: the interval in which a user creates WWW-traffic without being silent for more than a certain time. That is, a *session* starts when the first web page is fetched (the first request is made) and ends when the last page is received (but not yet read).

If we look at this on the application protocol level, then for each HTTP client the start of the first session is given by the transmission of the first request to a server. The server's response always belongs to the same session. If the next request occur within a certain time after the previous response is completed (or before it is completed) the new request and its response also belong to the same session. That is, two request-response pairs originated from the same client belongs to the same session if they are not separated by "to much time", an interval determined by a parameter here called $T_{session}$. If no request or response it sent for more than $T_{session}$ minutes then the next request is the start of a new session.

The packet trace doesn't contain application level HTTP requests and responses, but only lower level TCP/IP packet headers. Since the HTTP client sends nothing but requests, every TCP packet from a client - carrying some payload data (not pure acknowledgement or control packet) - is transferring a HTTP request. If the transferring of a TCP packet that carries a request is preceded by a period of $T_{session}$ minutes where no data is transferred to or from this client then (the timestamp of) this packet represents the start of a new session.

The problem is to determine a decent value of $T_{session}$. The value must be big enough, so the user is given time to read a web page without starting a new session, and small enough so that different periods of web surfing can be separated.

To characterize browsing strategies Catledge and Pitkow [5] used a version of Mosaic to log all user interaction with the browser. By calculating the mean and standard deviation of the time between each user interface event they determines that all events that occurred over 25.5 minutes apart should be delineated as a new session.

Vicari [30] used a *tcpdump* packet trace and tested different values ranging from 15 to 45 minutes, and the same sessions were found for each value. That made the choice of value less important.

With the trace used here, the value of $T_{session}$ do matter. Different values ranging from 10 to 45 minutes were tested and they all gave rise to a different number of sessions. With the value 10 minutes 645 sessions were found in the trace, if 15 minutes was chosen the trace contained 536 sessions etc. The higher the value of $T_{session}$ the fewer sessions were found. With $T_{session}$ = 45 minutes 299 sessions were found.

For each client the time of silence preceding the (second and all following) HTTP requests was calculated. In most cases this time interval is short, 54246 of 55252 requests was preceded by less than 5 minutes of silence. These short silences separates user clicks and the retrieval of different parts of a web page, not sessions. Figure 26 shows time of silence in 5 minute bins ranging from 5 to 50 minutes.

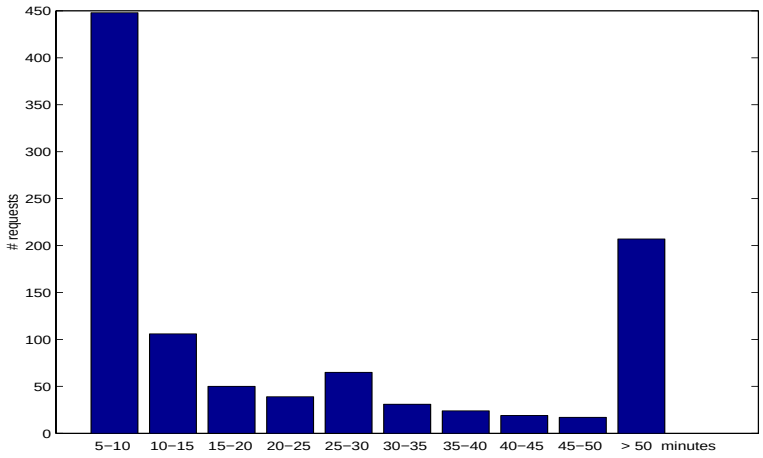

Figure 26. Time of silence preceding HTTP requests.

There were 1006 requests preceded by more than 5 minutes of silence. Of these requests 448 (44%) was preceded by less than 10 minutes of silence. The figure may suggest 10 minutes as a reasonable value, or at least as a lower limit. Otherwise the figure doesn't help much in the search for a value of $T_{session}$.

As stated above, the notion of a session is vague and hard to define. In the end it deals with how people behave when they surf the net, and a packet trace might not be the best tool for studying human behaviour. Ten minutes is not much time for a user to read the last received web page, stop surfing, get something else done and then return to the browser. Also, if a client uses the *Back* command on the browser to navigate, the page is fetched from the browsers cache, and no network traffic occur. This makes it even harder to determine session boundaries.

The value $T_{session}$ = 15 minutes was chosen.

4.2.4 User clicks

Mah [19] determines the number of files (HTML document and images) per web page, by investigation of a *tcpdump* packet trace. With the assumption that only HTTP/1.0 is used, there is one TCP connection for each file. He uses two simple heuristics to determine whether two connections belong to the same document. First, the two connections must originate from the same IP address and second, the two connections cannot be separated by too much time. If two connections are separated by more than a certain time then the user have clicked on a link and the connections belong to different pages.

Today, the assumption that only HTTP/1.0 is used, is not valid. The example in section 4.1.3 shows that some common web browsers give rise to persistent connection. There can be more than one HTTP request per TCP connection, and a connection can be held open after a user click to transfer more than one web page. This means that the time between connections cannot be used to determine if a user have clicked on a link to fetch a new web page. Instead only the time between the last HTTP response (or request) and a new request is considered, irrespective of which TCP connection the client uses for the transfer.

The reasoning here is the same as above when session boundaries were determined. Every TCP packet from a client that carries some payload data (not pure acknowledgement or control packet) is transferring a HTTP request. If the transfer of a TCP packet that carries a request is preceded by a period of T_{click} seconds where no data is transferred to or from this client then (the timestamp of) this packet represents a user click.

The problem is to determine the value of T_{click}. The value must be large enough, so that requests for parts of the same web page is not counted as user clicks, and small enough to separate different user clicks.

As described in the previous section, for each request the time of silence preceding it was calculated. From these times the requests were sorted and counted. Figure 27 shows the result with time of silence in 0.2 second bins ranging from 0 to 2.2 seconds.

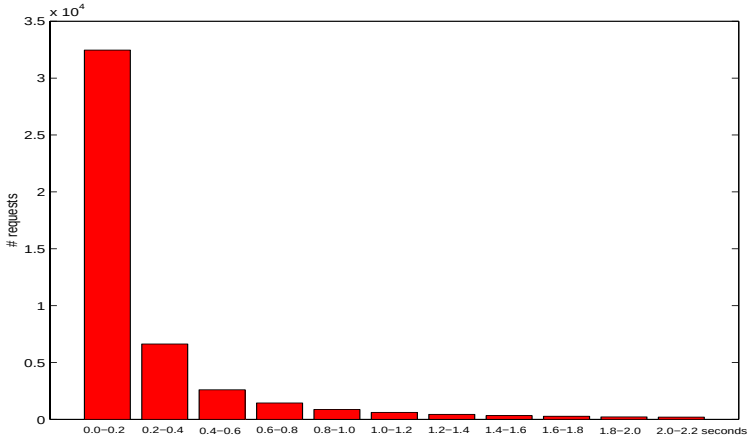

Figure 27. Time of silence preceding HTTP requests, in the range 0 to 2.2 seconds.

Mah [19] uses the threshold value 1 second to separate connections that belongs to different web pages. The main reason for the choice of this value was that users will generally take longer than one second to react to the display of a new page before they order a new document retrieval. When trying to find possible causes of self-similarity in WWW traffic, Crovella [8] investigates OFF times that correspond to periods when a workstation is not receiving web data. He concludes that OFF times in the range of 1 ms to 1 second is determined by machine processing and display time for data items that are retrieved, not due to users examining data.

From this the value T_{click} = 1 second was chosen, even though the figure above might suggest that an even smaller value would have been reasonable.

4.2.5 Tcpdump, Awk and Matlab

Awk was used to extract the needed information from the *tcpdump* file. The extracted data was later investigated further using *Matlab.* Below is an example of what a *tcpdump* file look like:

14:27:35.707978 193.10.65.225.4110 > 130.238.7.10.80: S 120339986:120339986(0) win 8192 <mss 1460> (DF)

14:27:35.718623 130.238.7.10.80 > 193.10.65.225.4110: S 207593203:207593203(0) ack 120339987 win 16384 <mss 512>

14:27:35.719135 193.10.65.225.4110 > 130.238.7.10.80: . ack 1 win 8192 (DF)

14:27:35.719765 193.10.65.225.4110 > 130.238.7.10.80: P 1:243(242) ack 1 win 8192 (DF)

14:27:35.756345 130.238.7.10.80 > 193.10.65.225.4110: . ack 243 win 16384

14:27:35.768221 130.238.7.10.80 > 193.10.65.225.4110: . 1:513(512) ack 243 win 16384

14:27:35.768410 130.238.7.10.80 > 193.10.65.225.4110: . 513:1025(512) ack 243 win 16384

.....

From this, only information about packets carrying payload data was extracted.The input to *Matlab* was a matrix where each such packet was represented with a timestamp in microseconds, a unique client id, direction (0 = client request, 1= server response), and the packet size without headers:

52055719765	1225	0	242
52055768221	1225	1	512

```
52055768410        1225        1        512
.....              ....        .        ...
```

A *Matlab* program was written which for each client went through the times between requests and responses and used $T_{session}$ and T_{click} to determine session lengths, time between user clicks within a session and the amount of data transferred as response to a user click. The sessions that started within $T_{session}$ minutes from the beginning of the trace, and sessions not finished when less than $T_{session}$ minutes remained was discarded.

4.3 Results

4.3.1 Model representation

There are two basic approaches to representing the probability distributions that make up a traffic model. One is to attempt to fit the observed data to well-known probability distributions that are described analytically in a simple mathematical form. This approach has the advantage of being compact, thus easily communicated, and also makes it easy to compare different datasets. The disadvantage is that datasets cannot always be well described by known distributions. The alternative is an empirical model that represent probability distributions by the observed cumulative distribution functions (CDF). This approach is obviously less compact, requires more storage, and makes it somewhat harder to compare different datasets, but can be used to represent arbitrary probability distributions. Here no attempt is made to fit the data to well-known distributions. Instead the more flexible CDF representation is used for all distributions. To generate values from a CDF, the inverse transformation method, for instance described in Jain [16], can be used.

4.3.2 Session lengths

There were a total of 536 complete sessions. The minimum session length was 0.049683 seconds and the maximum was 4 hours, 13 minutes and 13.6 seconds. The very short sessions appear when users do just one single click and then nothing more. That sometimes give rise to just one request and a short response, and altogether only a few hundredth of a second of network traffic. Figure 28 shows a histogram and Figure 29 the empirical CDF of session lengths.

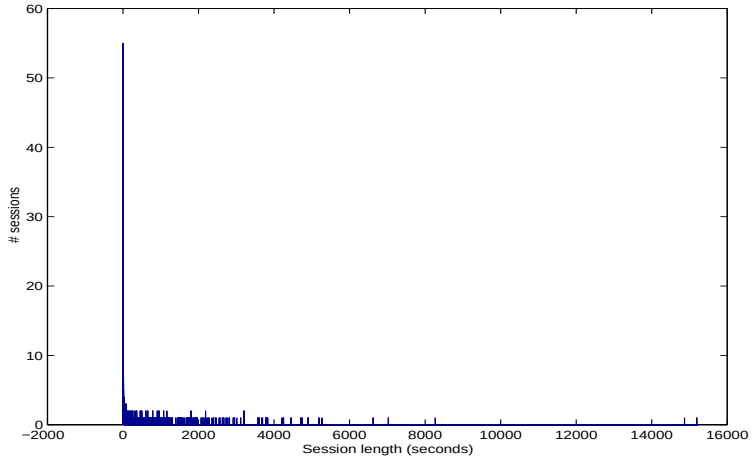

Figure 28. Histogram of session lengths.

The median session length was 239 seconds (3 minutes and 59 seconds) and the mean was 770 seconds (12 minutes and 50 seconds) with a standard deviation of 1421 seconds. The coefficient of variation was 1.8.

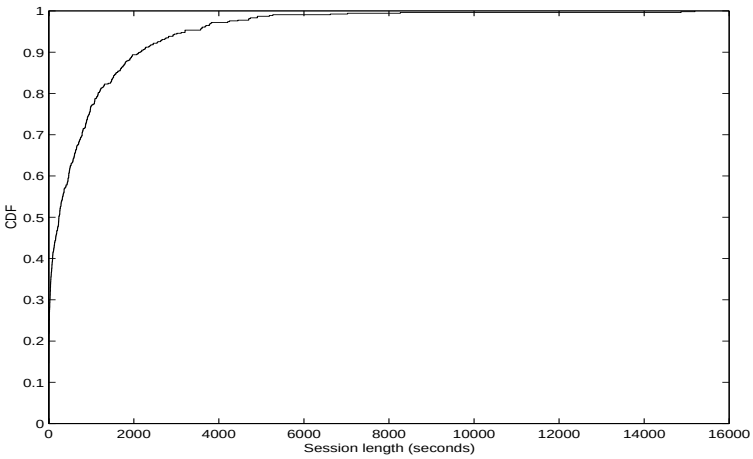

Figure 29. Empirical CDF of session lengths.

4.3.3 Interarrival times of user clicks

Below, a histogram and the CDF of interarrival times of user clicks are shown. These are the time interval between user clicks within a session. There are a total of 10088 interarrival times in the data set. The minimum time was 1.001317 seconds, just above the $T_{session}$ threshold of one second. The maximum time between two user clicks within the same session was 5989 seconds! That is, data was transferred in 1 hour 39 minutes and 49 seconds without interruption. Since only four values are above 1000 seconds the range in the figures below have the upper limit 1000 seconds.

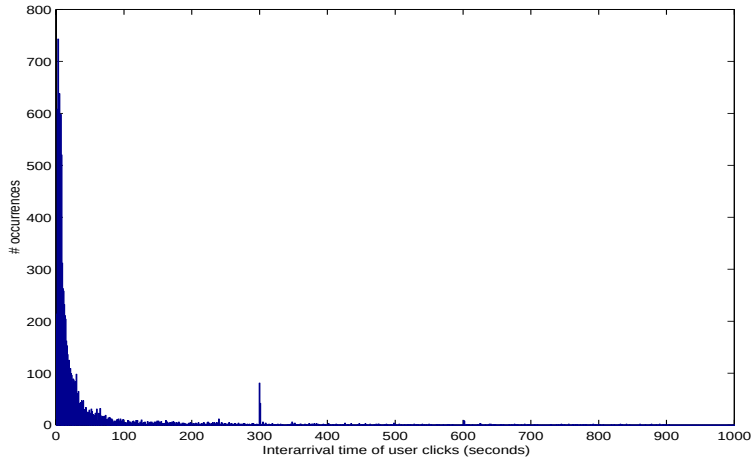

Figure 30. Histogram of interarrival times of user clicks.

The median interarrival time was 11 seconds and the mean 47 seconds with a standard deviation of 125 seconds. The coefficient of variation was 2.7.

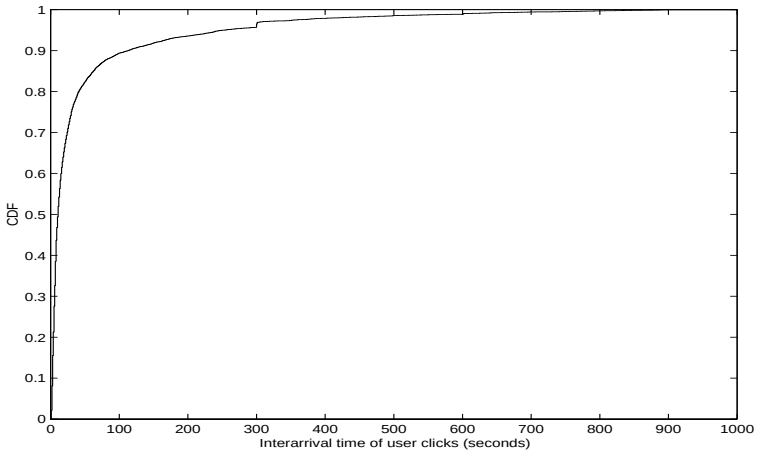

Figure 31. Empirical CDF of interarrival times of user clicks,

4.3.4 Data transferred

The amount of data transferred from servers as response to a single user click varies a lot. On one occasion 72707362 bytes were transferred and at other times no data at all was received by the client. There is no way to show a meaningful histogram that covers such a large range of values. Here only the part of the CDF that covers values below 250000 bytes is shown. Only 164 of the total 10624 values were larger than 250000.

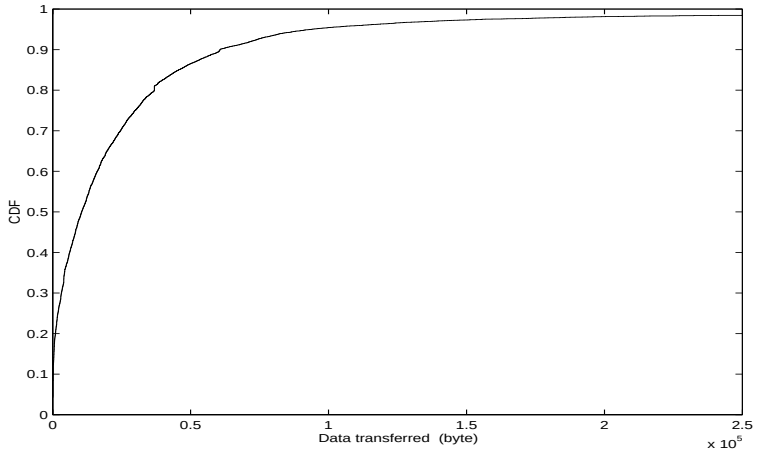

Figure 32. Empirical CDF of the amount of data transferred as response to a single user click.

The median was 10511 bytes and the mean was 54144 bytes with a standard deviation of 921851 bytes. The coefficient of variation was 17.0.

In the results presented above it shows that in some cases no data at all is received. Actually, more than 4% of the discovered user clicks doesn't get any response. This is somewhat peculiar and might indicate that something is wrong. No thorough investigation of all these cases is done here, but a closer examination of the packet trace for some of them shows the same result. Below is an extract from the packet trace:

....

11:39:20.450950 client.sics.se.1085 > server1.com.http: F 879:879(0) ack 17460 win 16926 (DF)

11:39:20.622081 server1.com.http > client.sics.se.1085: . ack 880 win 16926 (DF)

11:39:55.448682 client.sics.se.1087 > server2.se.http: S 453297664:453297664(0) win 16616 <mss 546,wscale 0,eol> (DF)

11:39:55.463524 server2.se.http > client.sics.se.1087: S 2053890870:2053890870(0) ack 453297665 win 1092 <mss 1460> (DF)

11:39:55.498164 client.sics.se.1087 > server2.se.http: P 1:391(390) ack 1 win 16926 (DF)

11:39:56.999251 client.sics.se.1087 > server2.se.http: P 1:391(390) ack 1 win 16926 (DF)

11:39:57.114233 server2.se.http > client.sics.se.1087: P 1:227(226) ack 391 win 8346 (DF)

....

The rows in *italic style* represents client requests. The first request is preceded by more than 35 seconds where no data is sent to or from this client so this clearly represents a user click. Following this request there is a silence of more than 1.5 seconds and then the same request appears again. Why? Is it a retransmission or a duplicate of the same packet? Are there packets missing? In either case, this second request is not due to a user clicking a link. But since it is preceded by more than T_{click} seconds of silence the algorithm used detects it as a new click, and so no data was transferred in response to the previous request.

4.3.5 Remarks

The results presented above can be used as a model of WWW session. The intention was to create a simple model, not the perfect one. A few weaknesses in the model and the methodology used were mentioned in the previous sections. Some additional shortcomings are to be presented here.

The threshold values $T_{session}$ and T_{click} are heuristic. Although a lot of effort were put into determining these values all cases cannot be covered. For instance, if a client quickly clicks to navigate to another web page before the transfer of the previous one was completed. Then this click will not be discovered.

The users are separated by IP addresses. If several clients uses the same machine they cannot be distinguished. This is probably unusual and thus of little consequence.

The choice of variables. Here session lengths, interarrival time of user clicks and the amount of data transferred as response to a user click, were studied. These are not the only three variables that can be used in order to model the traffic. For instance Mah [19], examines request and reply lengths, document sizes, user think time, consecutive document retrievals from the same server and the relative popularity of any server. Considering the problem of determining session boundaries it might not be necessary to use sessions at all. Instead, only examine the interarrival time of user clicks even though there might be days between them.

Dependencies between the variables studied. It would be interesting to analyse if there is a correlation between the interarrival time of user clicks and the amount of data transferred.

The presence of caching. Most clients employ caching to speed up access to WWW files. When a request is made, the browser software first checks to see if the file requested can be found in the cache. The file can then be copied directly from the cache instead of being retrieved over the Internet. The risk with caching is that it might return a response different from what would be returned by direct communication with the origin server. The HTTP protocols specify different headers, such as *Expires*, *Last-Modified* and *If-Modified-Since*, to facilitate the use of caching. Although, here only the actual traffic generated is of interest, the use of caching do influence the results. As mentioned in 4.2.3 the use of caching makes it harder to determine the value $T_{session}$. Also the amount of data transferred due to a user click depends on whether the files requested resides in cache or not. If the same web pages were requested using browsers with different caching mechanisms the results would differ.

In order to evaluate the model, traffic should be generated and compared to real traffic. For a meaningful simulation, the actual data transferred must be regulated by the TCP congestion an flow control mechanism, which are not included in this model. Doing this is beyond scope of this work.

5.0 Summary

In this thesis work, network traffic captured at SICS and on the Supernet was analysed. The external traffic at SICS was shown to be self-similar with the Hurst parameter estimated to $H \approx 0.8$. The value depending somewhat on estimation method, bin size and point of time. For the Supernet trace most estimates were above 0.8, so the traffic is clearly self-similar. The traffic was also examined with respect to which protocols and packet sizes are present and in what proportions. In the SICS trace most packets are small, TCP was shown to be the predominant transport protocol and NNTP the most common application. In contrast to this, large UDP packets sent between not well-known ports dominates the Supernet traffic. Finally, characteristics of the client side of

the WWW traffic was examined more closely in order to create a simple model of WWW-sessions. Empirical probability distributions were derived describing session lengths, time between user clicks and the amount of data transferred due to a single user click.

References:

[1] P. Barford, A. Bestavros, A. Bradley and M. Crovella, “Changes in web client access patterns: characteristics and caching implications,” to appear in *World Wide Web, Special issue on characterization and performance evaluation*, 1999.

[2] P. Brockwell and R. Davis. *Time Series: Theory and Methods*. Springer-Verlag, New York, 1987.

[3] P. Barford and M. Crovella, “A performance evaluation of hyper text transfer protocols,” in *Proceedings of ACM SIGMETRICS ‘99*, Atlanta, Georgia, May 1999.

[4] J. Beran. *Statistics for Long-Memory Processes*. Chapman & Hall, New York, 1994.

[5] L. D. Catledge and J. E. Pitkow, “Characterizing browsing strategies in the World-Wide Web,” in *Proceedings of the third International World Wide Web Conference*, Darmstadt, Germany, April 1995.

[6] K. Claffy, G. Miller and K. Thompson, “The nature of the beast: recent traffic measurements from an Internet backbone” at http:// www.caida.org/Papers/Inet98.

[7] C. R. Cunha, A. Bestavros, and M. E. Crovella, “Characteristics of WWW client-based traces,” Technical Report BU-CS-95-010, Computer Science Department, Boston University, July 1995.

[8] M. E. Crovella and A. Bestavros, “Self-similarity in World Wide Web traffic: evidence and possible causes,” in *IEEE/ACM Transaction on Networking*, Vol. 5, no. 6, pp.835-846,1997.

[9] M.E. Crovella, M. S. Taqqu, and A. Bestavros, “Heavy-Tailed Probability Distributions in the World Wide Web,” in *A practical guide to heavy tails: statistical techniques and applications*, R. Adler, R. Feldman and M.S. Taqqu, Eds. Birkhauser, Boston, 1998.

[10] A. Erramilli, O. Narayan, and W. Willinger, “Experimental queueing analysis with long-range dependent packet traffic,” in *IEEE/ACM Transactions on Networking*, Vol. 4, no. 2, pp. 209-223, 1996.

[11] A. Feldmann, A.C. Gilbert, W. Willinger and T.G. Kurtz, “The changing nature of network traffic: Scaling phenomena,” in *Computer Communication Review* 28, no.2, April 1998.

[12] A. Feldmann, A.C. Gilbert, and W. Willinger, “Data networks as cascades: Investigatingthe multifractal nature of Internet WAN traffic,” in *Computer Communication Review* 28, no. 4, pp. 42-55, October 1998.

[13] M. Grossglauser and J-C. Bolot, “On the relevance of long-range dependence in network traffic,” in *Proceedings of SIGCOMM’96*, pp.15-24.

[14] The Internet Traffic Archive. Available at http://www.acm.org/sigcomm/ITA .

[15] V. Jacobson, C. Leres and S. McCanne, tcpdump software. This software is available at ftp://ftp.ee.lbl.gov/tcpdump.tar.Z .

[16] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, New York, 1991.

[17] B. Krishnamurthy, J. C. Mogul and D. M. Kristol, “Key differences between HTTP/1.0 and HTTP/1.1,” in *Computer Networks*, vol. 31, no. 11-16, pp 1737-1751, 1999.

[18] W. E. Leland, M. S. Taqqu, W. Willinger and D. V. Wilson, “On the self-similar nature of Ethernet traffic (Extended version),” *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 1-15, 1994.

[19] B. A. Mah, “An empirical Model of HTTP network traffic,” in *INFOCOM ‘97 Conference Proceedings*, pp. 592-600, Kobe, Japan april 7-11, 1997.

[20] S. Molnar, A. Vidacs and A. Nilsson, “Bottlenecks on the Way Towards Fractal Characterization of Network Traffic: Estimation and Interpretation of the Hurst Parameter”. International Conference on the Performance and Management of Communication Network, Tsukuba, Japan, 17-21 November 1997.

[21] V. Paxson and S.Floyd, “Wide-area traffic: The failure of poisson modeling,” in *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 226-244, 1995.

[22] V. Paxson and S. Floyd, “Why we don’t know how to simulate the Internet,” in *Proceedings of the 1997 Winter Simulation Conference*, Atlanta, GA, 1997.

[23] RFC 1945 *Hypertext Transfer Protocol -- HTTP/1.0*, available at http://www.ietf.org/rfc/rfc1945.txt

[24] RFC 2616 *Hypertext Transfer Protocol -- HTTP/1.1*, available at http://www.ietf.org/rfc/rfc2616.txt

[25] W. R. Stevens, *TCP/IP Illustrated, Vol. 1: The Protocols*, Addison-Wesley, Reading, 1994.

[26] W. R. Stevens, *TCP/IP Illustrated, Vol. 3: TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols*, Addison-Wesley, Reading, 1996.

[27] M. S. Taqqu, V. Teverosky, W. Willinger, “Estimators for long-range dependence: an empirical study,” in *Fractals*, vol. 3, no. 4, pp. 785-788, 1995.

[28] M. S. Taqqu and V. Teverovsky, “On estimating the intensity of long-range dependence in finite and infinite variance time series,” in *A practical guide to heavy tails: statistical techniques and applications*, R. Adler, R. Feldman and M.S. Taqqu, Eds. Birkhauser, Boston, 1998.

[29] K. Thompson, G. J. Miller and R. Wilder, “Wide-area traffic patterns and characteristics (Extended version),” in *IEEE Network*, vol. 11, no. 6, pp. 10-23, 1997.

[30] N. Vicari, “Measurement and Modeling of WWW-Sessions”, Research Report Series, Instutite of Computer Science,University of Wurzburg, 1997.

[31] W. Willinger, M. S. Taqqu, R. Sherman, and D. Wilson, “Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level,” in *Proceedings of SIGCOMM ‘95*, pp. 100-113.

[32] W. Willinger and V. Paxson, “Where Mathematics meets the Internet,” in *Notices of the AMS*, vol. 45, no. 8, pp. 961-970, 1998.

[33] W. Willinger, V. Paxson and M.S. Taqqu, “Self-similarity and heavy tails: structural modeling of network traffic,” in *A practical guide to heavy tails: statistical techniques and applications*, R. Adler, R. Feldman and M.S. Taqqu, Eds. Birkhauser, Boston, 1998.

[34] A. Woodruff, P. M. Aoki, E. Brewer, P. Gauthier, and L. A. Rowe, “An investigation of documents from the World Wide Web,” in *Proceedings of the Fifth International World Wide Web Conference*, Paris, France, May 1996.